
Ecosystem for COllaborative Manufacturing PrOceSses – Intra- and Interfactory Integration and AutomaTION
(Grant Agreement No 723145)

# D4.4 Prototype of the Security Framework I

## Date: 2018-07-12

## Version 1.1

**Published by the COMPOSITION Consortium**

**Dissemination Level: Public**

Co-funded by the European Union's Horizon 2020 Framework Programme for Research and Innovation under Grant Agreement No 723145

# Document control page

**Document file:** D4.4 Prototype of the Security Framework 1_v1.1.docx
**Document version:** 1.1
**Document owner:** ATOS

**Work package:** WP4 – Secure Data Management and Exchange in Manufacturing
**Task**: T4.4 – Cyber Security for Factories
**Deliverable type:** OTHER

**Document status:** ☒ Approved by the document owner for internal review
☒ Approved for submission to the EC

**Document history:**

| Version | Author(s) | Date | Summary of changes made |
|---|---|---|---|
| 0.1 | Nacho González, Rodrigo Díaz, David Rojo, Mario Faiella (ATOS) | 2018-06-15 | First version of the deliverable. |
| 0.2 | Nacho González, Rodrigo Díaz, David Rojo, Mario Faiella (ATOS) | 2018-06-22 | Configuration and common chapters. |
| 0.3 | Nacho González, Rodrigo Díaz, David Rojo, Mario Faiella (ATOS) | 2018-06-29 | Reputation model and EPICA integration. Ready for internal quality process. |
| 0.4 | Nacho González, Rodrigo Díaz, David Rojo, Mario Faiella (ATOS) | 2018-07-05 | Internal review comments addressed. |
| 0.5 | Nacho González, Rodrigo Díaz, David Rojo, Mario Faiella (ATOS) | 2018-07-07 | Addressed minor typos and wrong references |
| 0.6 | Nacho González, Rodrigo Díaz, David Rojo, Mario Faiella (ATOS) | 2018-07-10 | Final version edited and approved by document owner |
| 1.0 | Helene Udsen (IN-JET) | 2018-07-10 | Document amended and edited to align with COMPOSITION Template<br>Internal reviewer comments reinserted<br>Final version submitted to the European Commission |
| 1.1 | Nacho González (ATOS) | 2018-07-12 | |

**Internal review history:**

| Reviewed by | Date | Summary of comments |
|---|---|---|
| Matteo Pardi (NXW) | 2018-07-03 | Approved subject to addressing of several suggestions for improvement |
| Luis Martins (BSL) | 2018-07-04 | Approved with comments and suggestions |

# Index:

# 1 Abbreviations and Acronyms

| Acronym | Meaning |
|---|---|
| **OTP** | One Time Password |
| **SSL** | Secure Sockets Layer |
| **HTTPS** | Hyper Text Transfer Protocol Secure |
| **REST** | Respresentational State Transfer |
| **RaaS** | RabbitMQ authentication and authorization Service |
| **PAP** | Policy Administration Point |
| **XACML** | eXtensible Access Control Markup Language |
| **PA** | Policy Administrator |
| **PG** | Policy Generator |
| **PV** | Policy Validator |
| **PH** | Policy Handler |
| **PEP** | Policy Enforcement Point |
| **PIP** | Policy Information Point |
| **PDP** | Policy Decision Point |
| **PRP** | Policy Retrieval Point |
| **WM** | Weighted Mean |
| **OWA** | Ordered Weighted Aggregation |
| **WOWA** | Weighted Ordered Weighted Aggregation |
| **HWM** | Hybrid Weighted Mean |
| **AM** | Arithmetic Mean |

# 2 Introduction

## 2.1 Purpose, context and scope of this deliverable

This document provides some guidelines about how the specifications and requirements of the Security Framework described in previous deliverables have been implemented and deployed. These previous deliverables are:

- *D4.1 Design of the Security Framework I* (delivered in M12)
- *D4.2 Design of the Security Framework II* (delivered in M18)

Furthermore, technical descriptions (basically regarding components integration and deployment) and other support information such as typical management operations, common configurations… have been included in this deliverable.

The first version of this Security Framework prototype provides a first deployment of the different component security services divided into different areas: authentication services, authorization services and cybersecurity. For each area, the mechanisms described in previous deliverables have been implemented and deployed as a first attempt. Later on, these components will be deployed in the final COMPOSITION production environment.

## 2.2 Content and structure of this deliverable

This document is structured in different sections detailing each one of the most representative areas, functionalities and cores of the Security Framework:

- **General Security Framework architecture**: how the proposed reference architecture has been implemented in this very first prototype.
- **COMPOSITION Reputation Model**: documentation about the reputation model, this is how agents of the marketplace scenario make trust-based decisions.
- **Deployment**: how the components are installed and deployed.

# 3 General Security Framework architecture

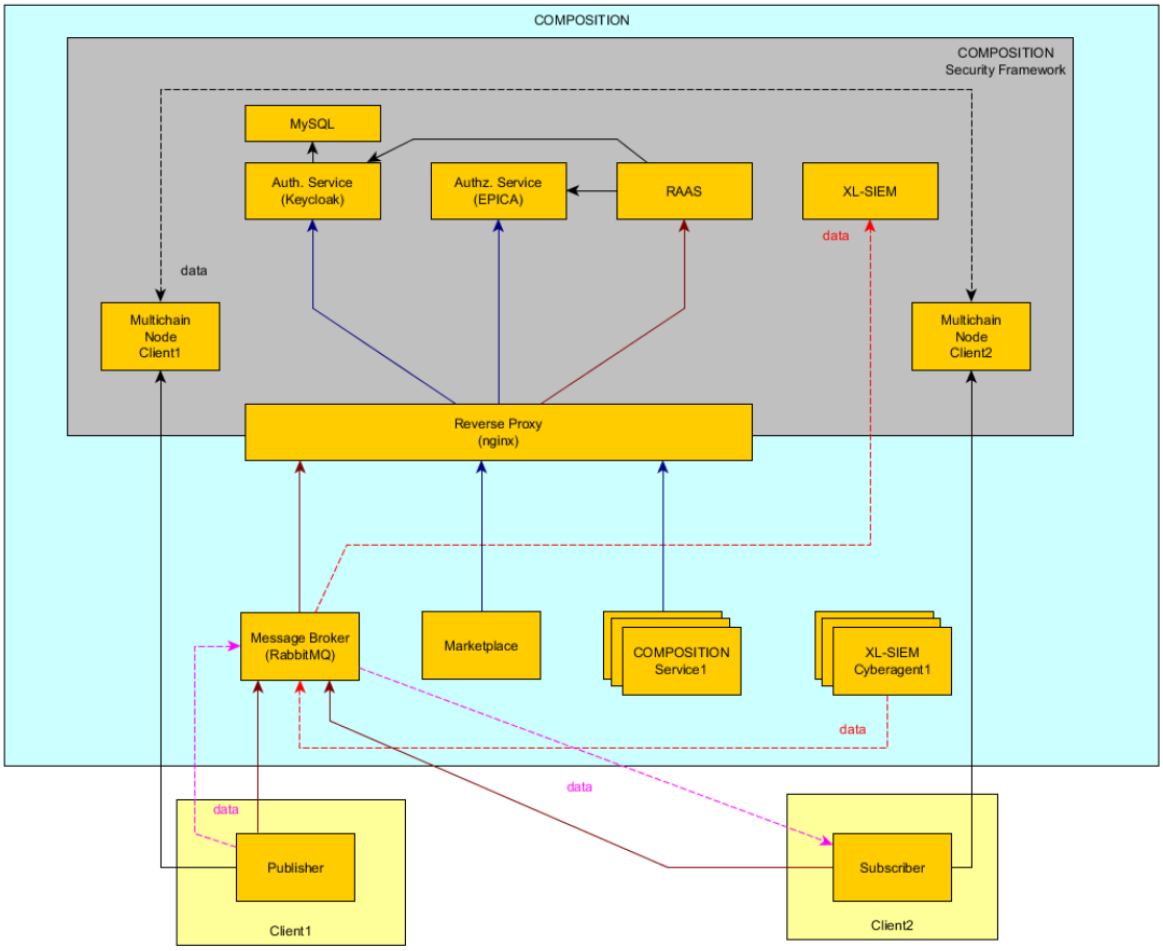


**Figure 1 Security Framework architectural design**

The architecture model followed in the implementation of the first prototype of the Security Framework is widely described in previous deliverables such as:

- *D4.1 Design of the Security Framework I*
- *D4.2 Design of the Security Framework II*

A brief description of the different components deployed:

- Authentication service (Keycloak)

  The main task of this service is providing the authentication mechanisms for users, applications, services and devices. The following standard authentication protocols are supported by Keycloak (OAuth[1], Open ID Connect[2] and SAML 2.0[3])

- Authorization service (EPICA)

  This component is responsible for providing authorization mechanisms to other COMPOSITION components. It is based on XACML v3.0 which provides an attribute-based access control mechanism and provides the means to define authorization policies used to protect resources.

- RAAS (RabbitMQ authentication and authorization service)

[1] https://oauth.net/2/
[2] http://openid.net/connect/
[3] https://en.wikipedia.org/wiki/SAML_2.0

This component in development is an http service whose main task is enabling the use of the Authentication (Keycloak) and Authorization (EPICA) services by the Message Broker (RabbitMQ).

- Reverse proxy

  This component is responsible for directing client requests to the appropriate backend server and also securing communication by enabling the use of TLS[4] (Transport Layer Security) cryptographic protocol. TLS provides security over a computer network, and aims primarily to provide privacy and data integrity between two communicating applications.

- Public Key Infrastructure (PKI)

  All messages flowing in the COMPOSITION platform trough the COMPOSITION Message Broker (RabbitMQ) are signed using JWS[5] (JSON Web Signature) standard proposed by IETF[6].

- Message logging

  Log management of all the messages sent trough the platform.

- Reputation model

  Reputation management of the stakeholders and the way to share it with other stakeholders.

## 3.1 Authorization Service (EPICA)

Differently from the other components of the Security Framework, few details have been provided about EPICA in previous deliverables *D4.1 Design of the Security Framework I* and *D4.2 Design of the Security Framework II*. Indeed, the main objective of this section is to describe the EPICA architecture, together with its main functionalities.

EPICA is an Attribute Based Access Control (ABAC) tool based on XACML v3.0[7]. The main idea of EPICA is to provide the means to manage access control policies and then, evaluate incoming requests against these policies, when specific accesses should be granted. More specifically, the role of EPICA in COMPOSITION is to provide the Authorization service for the Security Framework. A high-level view of the component was already given in D4.1, and the related architecture is depicted in Figure 2.

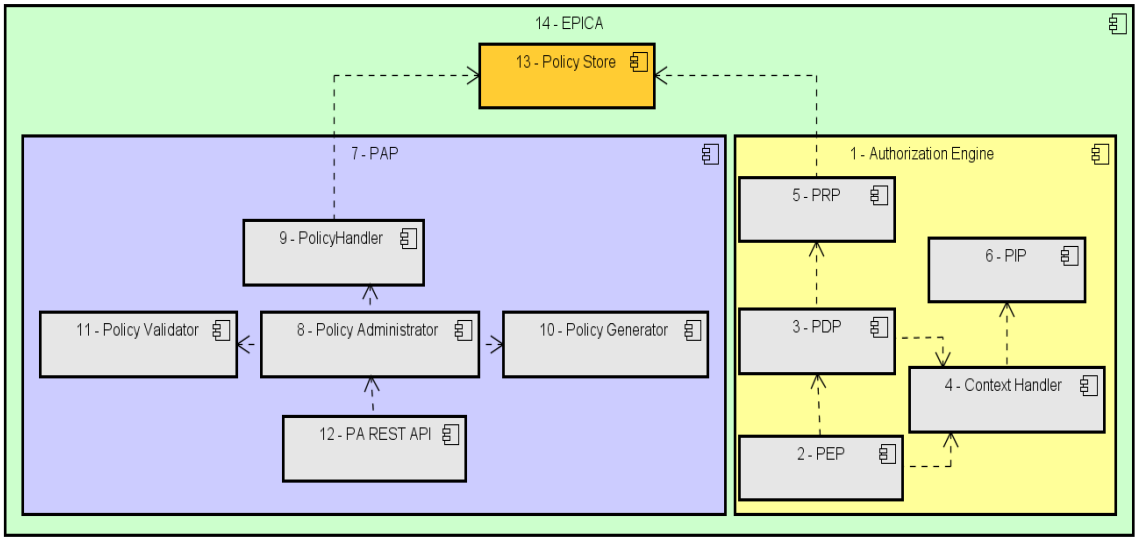


**Figure 2 EPICA Architecture**

[4] https://tools.ietf.org/html/rfc5246
[5] http://openid.net/specs/draft-jones-json-web-signature-04.html
[6] https://www.ietf.org/
[7] http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

### 3.1.1 Policy Administration Point (PAP)

PAP is in charge of managing policies that are used to check if specific permissions could be granted. This is achieved through a RESTful API, which provides different methods for handling access control policies. The workflow of the PAP is shown in Figure 3, where five main sub-components can be identified:

- **Policy Administrator RESTful API (PA REST API):** Policy Administration process is started through this component. It is characterised by a set of web services for accepting requests, to create, retrieve, update and delete access control policies. The details about the provided set of APIs will be described later.
- **Policy Administrator (PA):** this is the main component of the PAP and the backend component for the PA REST-API. It handles the requests to manage policies, orchestrating the rest of the components to realize each operation. Indeed, it contacts the Policy Generator to generate policies, the Policy Validator for validation and, finally, the Policy Handler to store them.
- **Policy Generator (PG):** it generates XACML policies in an automatic way, considering as input a JSON, where the parameters to be included in the policies are specified. The PG can be configured for working in a lightweight mode, making a lot easier the integration between the PA REST- API with other existing systems. In this way, the administrators do not need to have a complete knowledge of the XACML v3 standard, to manage policies. The PG follows the Factory software design pattern[8] and thus, purpose-specific generators can be created, depending on the context where EPICA is applied. These can act as parsers between the new input format (e.g. other than JSON) and the main generator, in charge to generate the final policy, or can be used to add some information (e.g., restrictions to all the policies generated).
- **Policy Validator (PV): its** purpose is to validate the format of the policy, more precisely if it is compliant to the XACML structure, before storing it. This is an essential check, to prevent any issues with the Policy Store; indeed, a single wrong policy might compromise the security of the protected resources. PV has a limited scope for now, it simply checks if the policy is a valid XML file, compliant with XACML directives. In the future, it could be extended to perform more sophisticated verifications on the policy, for instance semantic checks to verify specific statements.
- **Policy Handler (PH):** The Policy Handler is in charge of three main tasks: deploying, removing and updating existing policies. The PH is the component of the PAP which directly interacts with the Policy Store, thus it will need to adapt to the specific type of Policy Store used in the considered context. For this reason, the Factory software design pattern has been used for its implementation.

Summarizing, the Policy Administration process is started through the PA REST-API component. The list of the provided APIs is shown in Figure 4, generated through Swagger UI[9], while each method is briefly described in Table 1.

EPICA can search for policies in a hierarchical way when an access request is received, and this is very useful when the local filesystem is used for storing them (i.e., if a user requests an access to the sub-folder A-1, but A-1 does not have a policy associated to himself, EPICA will search in its parent folder A, and it will use the latter's policy, if exists). For this reason, EPICA must be able to provide APIs for handling policies at different levels. This feature could be disabled, if not needed.

[8] https://en.wikipedia.org/wiki/Factory_method_pattern
[9] https://swagger.io/

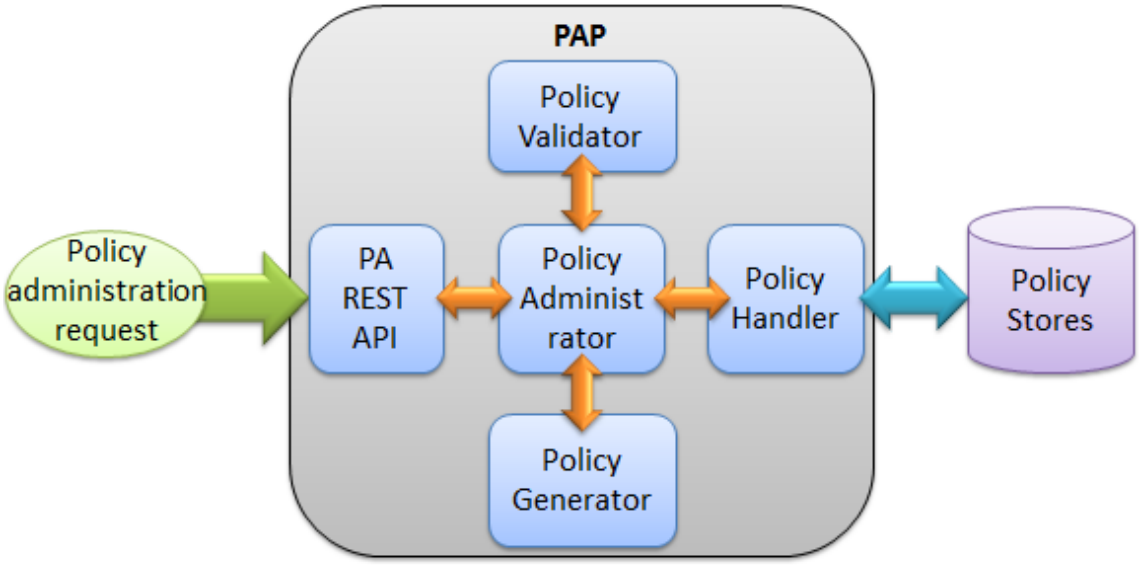


**Figure 3 PAP Workflow**

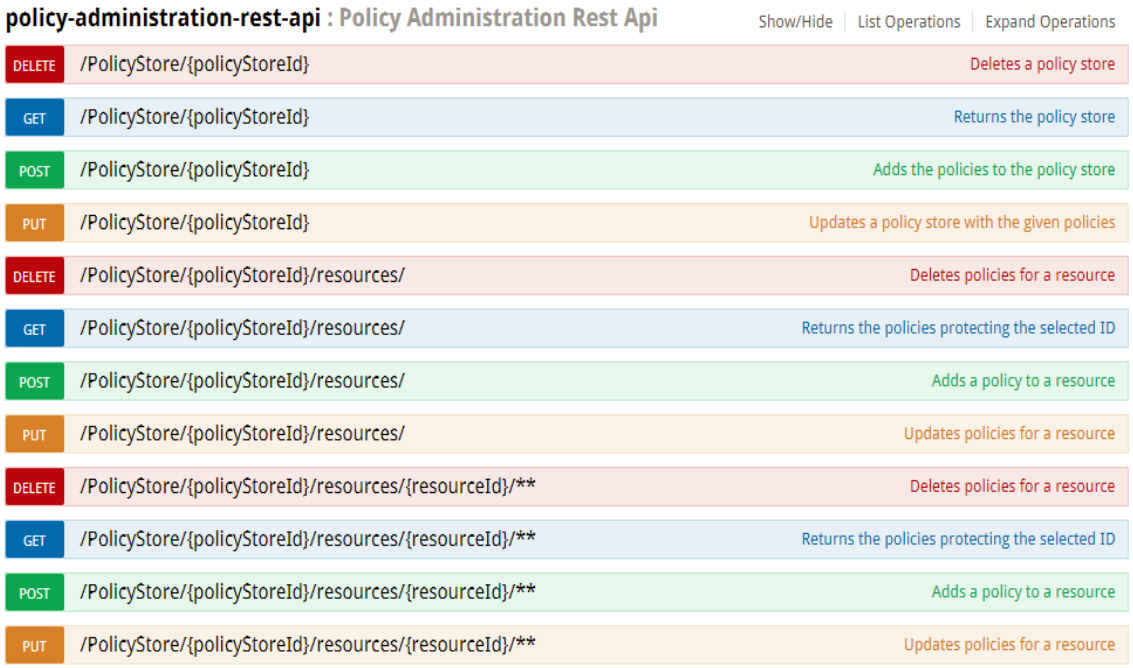


**Figure 4 PAP REST API**

**Table 1 PAP REST API Description**

| **/PolicyStore/{policyStoreId}** | | |
|---|---|---|
| Method | Path Parameters | Description |
| DELETE | policyStoreId: the ID of the policy store | Deletes the Policy Store |
| GET | policyStoreId: the ID of the policy store | Retrieve the Policy Store |
| POST | policyStoreId: the ID of the policy store | Creates a new Policy Store. Will not work if the Policy Store already exists |
| PUT | policyStoreId: the ID of the policy store | Updates the Policy Store, adding one or more policies to a store |

| **/PolicyStore/{policyStoreId}/resources/** | | |
|---|---|---|
| Method | Path Parameters | Description |
| DELETE | policyStoreId: the ID of the policy store | Deletes all the policies associated with the parent folder of the specified Policy Store |
| GET | policyStoreId: the ID of the policy store | Obtains all the policies associated with the parent folder of the specified Policy Store |
| POST | policyStoreId: the ID of the policy store | Creates one or more policies for the parent folder of the specified Policy Store |
| PUT | policyStoreId: the ID of the policy store | Updates one or more policies for the parent folder of the specified Policy Store |
| **/PolicyStore/{policyStoreId}/resources/{resourceId}/**** | | |
| Method | Path Parameters | Description |
| DELETE | policyStoreId: the ID of the policy store<br>resourceId: the ID of the selected resource | Deletes all the policies associated with the resourceId (i.e., a specific sub-folder of the specified Policy Store) |
| GET | policyStoreId: the ID of the policy store<br>resourceId: the ID of the selected resource | Obtains all the policies associated with the resourceId (i.e., a specific sub-folder of the specified Policy Store) |
| POST | policyStoreId: the ID of the policy store<br>resourceId: the ID of the selected resource | Creates one or more policies for the resourceId (i.e., a specific sub-folder of the specified Policy Store) |
| PUT | policyStoreId: the ID of the policy store<br>resourceId: the ID of the selected resource | Updates one or more policies for the resourceId (i.e., a specific sub-folder of the specified Policy Store) |

Depending on the specific type of the incoming HTTP request, the PA will handle it accordingly (i.e., if the request is related to the generation of a new policy, it will interact first with the PG for the policy generation, then with the PV for the validation, and, finally, with the PH for storing the policy in the Policy Store).

The main objective of the Swagger UI is to provide documentation about the APIs provided by the component; moreover, it could also be used for testing purposes, as can be seen in Figure 5, where the API for creating a policy for a specific resource is considered.

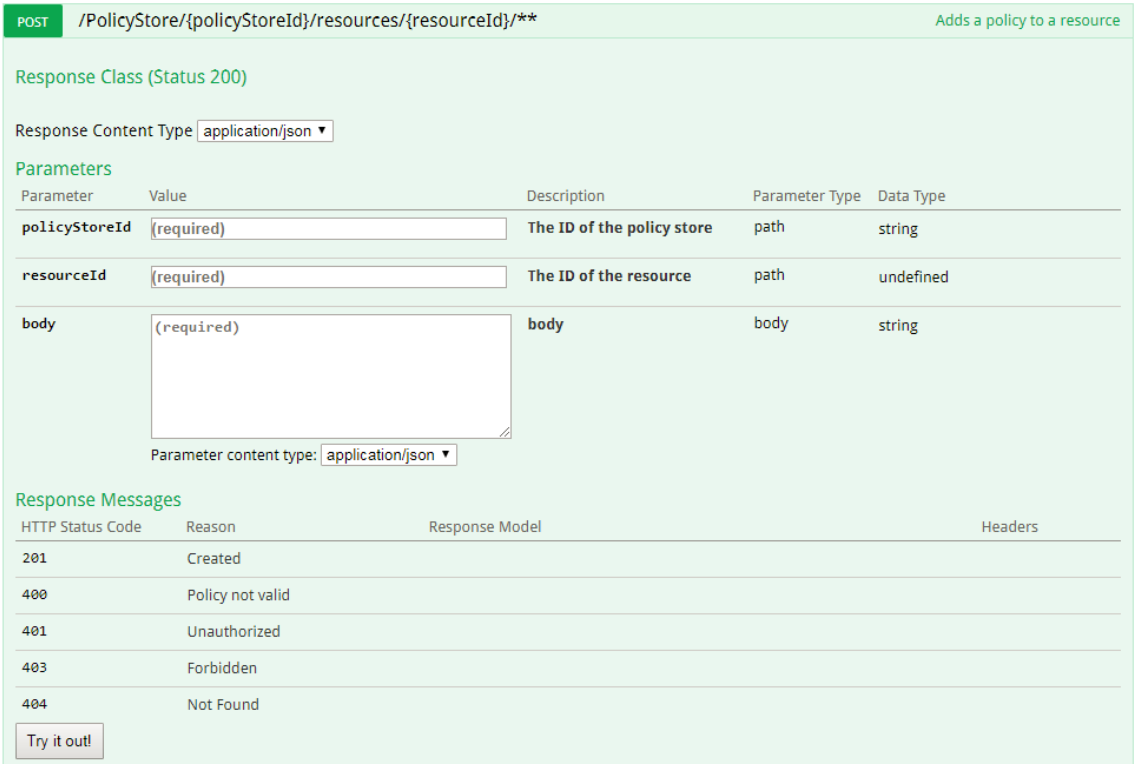


**Figure 5 Policy Test Creation**

More in details, following information are displayed:

- The HTTP method and the path for using the API, including the parameters (“policyStoreId” and “resourceId” in this case). If the local filesystem is used for storing policies, the resourceId is equal to the path where the policy will be stored, starting from the root folder of the specified Policy Store.
- API brief description.
- The parameters used by the method and the body, where the information needed by EPICA for creating the policy must be specified. A description for each of them, along with the corresponding text box, used for testing purposes, is also present.
- The response messages, and its reason/meaning, returned by the method itself.

After filling the text boxes, the test could be run clicking in the “Try it out!” button. The obtained response is shown in Figure 6 (where the IP address has been anonymised).

Request URL

```
http://172.18.0.6:8080/PolicyAdministrationAPI/PolicyStore/atos/resources/public/**
```

Response Body

```
no content
```

Response Code

```
201
```

Response Headers

```
{
  "date": "Fri, 22 Jun 2018 05:23:39 GMT",
  "transfer-encoding": "chunked",
  "content-type": null
}
```

**Figure 6 Policy Test Response**

The policy has been stored correctly, as confirmed by the Response Code, in the “public” sub-folder of the Policy Store characterised by the root folder called “atos”.

### 3.1.2 Authorization Engine

The Authorization Engine is the core component of EPICA, based on the XACML standard, in charge to deal with policy enforcement. That is, the process of evaluating an incoming request, against the related policy, for requesting the access to a specific resource, or more generically a permission, and to decide either to allow or deny the access/permission. The workflow of the Authorization Engine is shown in Figure 7.

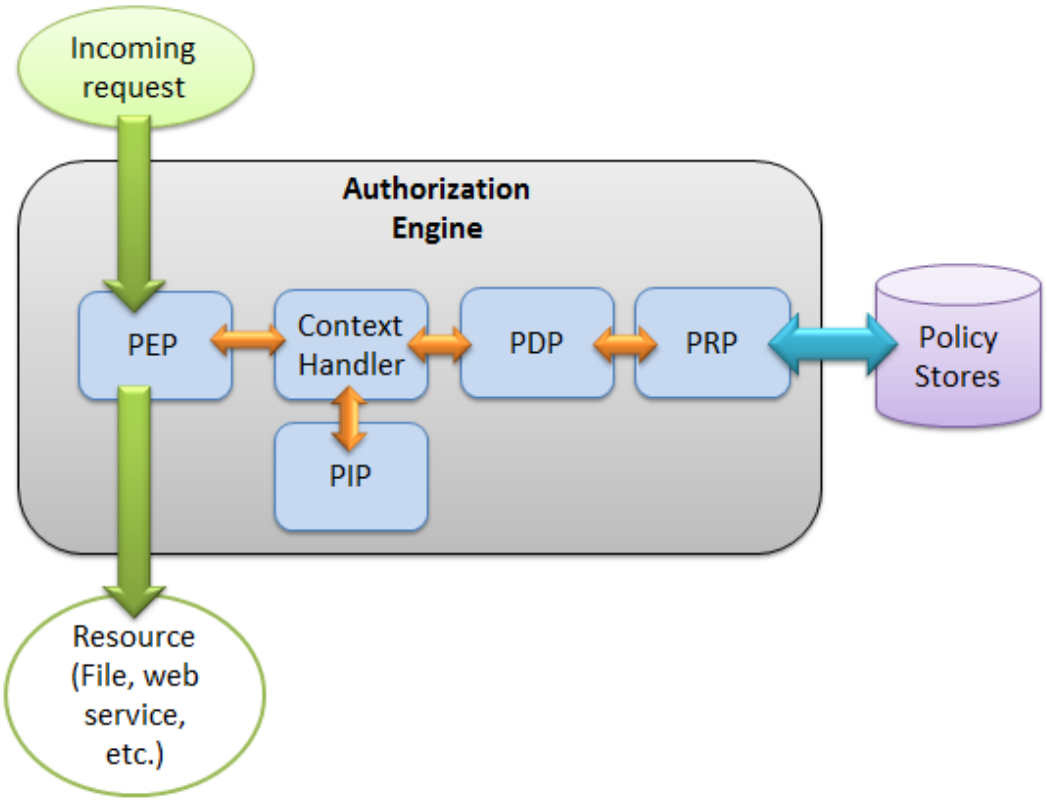


**Figure 7 Authorization Engine Workflow**

It is characterised by the following main components:

- **Policy Enforcement Point (PEP):** this component is the entry point of the Authorization Engine, and it is in charge of doing the enforcement. It will receive the request, and then contact the Context Handler for starting the evaluation process, and, finally, enforce the result of the evaluation. The PEP implementation used in COMPOSITION is able to receive HTTP requests through a REST API. The details will be given later.
- **Context Handler: its** main function is to extract, from the request received by the PEP, the information needed to build the XACML request, and, then, deliver it to the PDP. Additionally, it could also add specific attributes to the request itself, such as information obtained from the PIPs, if any.
- **Policy Information Point (PIP):** sometimes the incoming request does not contain all the needed information for its evaluation. In these cases, one or more PIPs can be used for obtaining this additional information. They can communicate with external components, called Attribute Managers (e.g., external databases, antiviruses, web services), from which they retrieve the attributes to be inserted in the request. In COMPOSTION, it is not planned to use any PIP, considering that all the information needed for the request evaluation should be extracted directly from the Keycloack[10] token present in the request itself.
- **Policy Decision Point (PDP):** one of the core components of the Authorization Engine. Its main task is to evaluate the incoming XACML request against the stored XACML policy, retrieved by the PRP from the Policy Store. The final decision will be sent back to the PEP, as a XACML response. The actual PDP used in EPICA is relying on the WSO2 Balana Open Source implementation[11], which supports XACML v3 standard.
- **Policy Retrieval Point (PRP):** this component will interact directly with the Policy Store, when specific policies should be retrieved for the access request evaluation. It has been implemented considering the Factory software design pattern, because it is strictly dependent on the chosen Policy Store, indeed adaptability has been taken into account. The obtained policies will be sent to the PDP, for performing the evaluation.

For a better understanding of the overall policy enforcement process, the sequence diagram in Figure 8 can be consulted.

As explained in D4.2, in COMPOSITION at least two Message Brokers (RabbitMQ[12]) will be implemented, for the Inter-Factory and the Intra-Factory scenarios, respectively. For this reason, the same number of RabbitMQ Authentication and Authorization Services (RaaS) needs to be deployed. EPICA, more precisely the PEP, will receive access requests directly from the RaaS, responding with the result of the policy enforcement process. It provides a REST API, shown in Figure 9 through the Swagger UI, and the related functionalities are described in Table 2. The Path parameters are, temporarily, used for inferring where the policies have been stored in the local filesystem, considering the specific entity that is making the request, while the Authorization field in the request Header will specify the authentication token. The Authorization Header, instead, will be used for receiving the Keycloack Authentication token.

This section concludes the part related to the description of EPICA main functionalities, and, in Section 5.1, it will be shown how it has been deployed. While, the implementation of the first integration with the RaaS will be described in Section 5.3, where a simple test case will also be presented, through the Swagger UI.

[10] https://www.keycloak.org/
[11] https://github.com/wso2/balana
[12] https://www.rabbitmq.com/

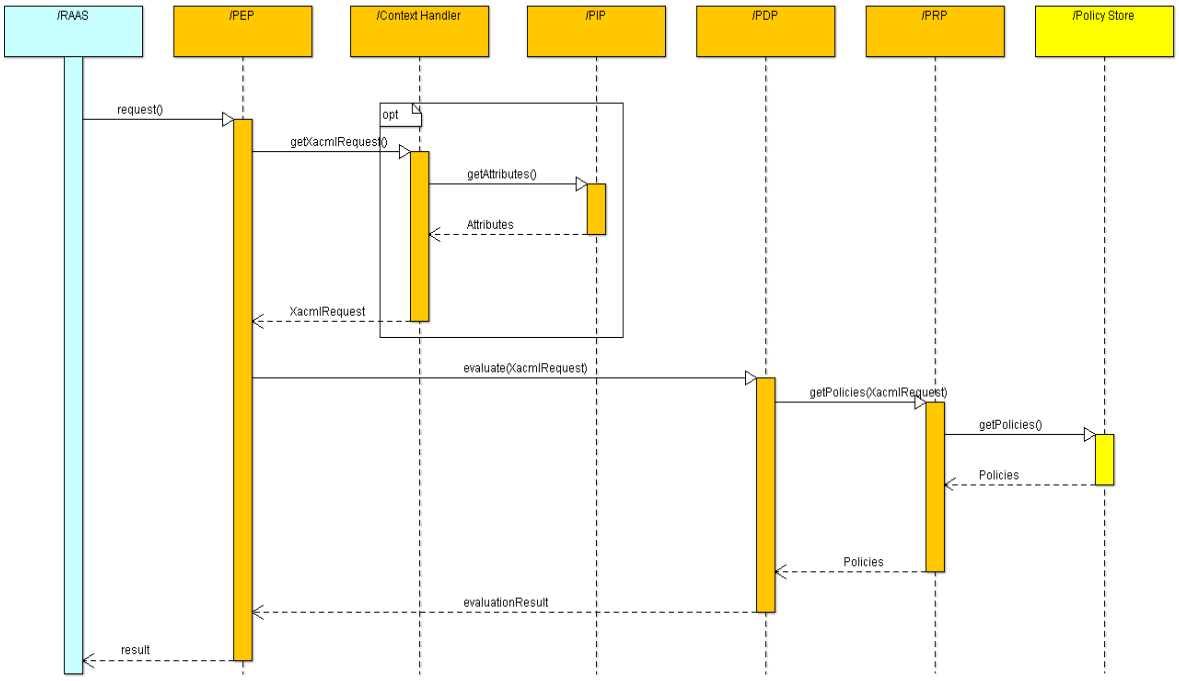


**Figure 8 Policy Enforcement Process**

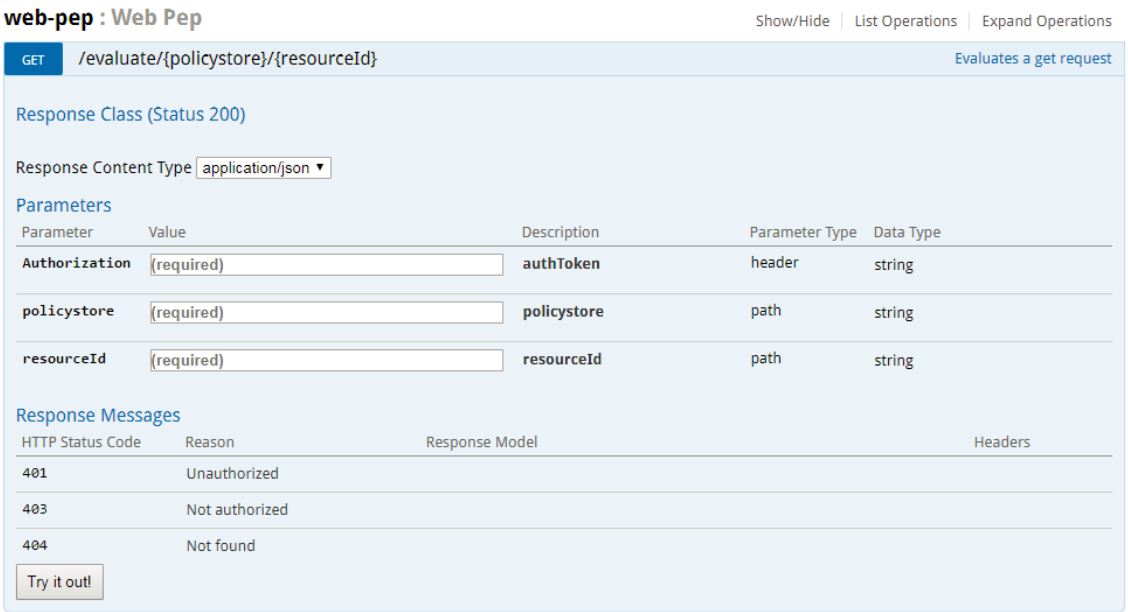


**Figure 9 Authorization Engine REST API**

**Table 2 Authorization Engine REST API Description**

| **/evaluate/{policystore}/{resourceId}** | | | |
|---|---|---|---|
| Method | Path Parameters | Header Parameters | Description |
| GET | policystore: the ID of the policy store<br>resourceId: the ID of the selected resource | Authorization: authentication token | Start the policy enforcement process, for evaluating the received access request |

# 4 COMPOSITION Reputation Model

In the COMPOSITION marketplace scenario, two different categories of agents are involved, (i) Marketplace agents and (ii) Stakeholder agents. Their features and roles have been described in *D2.3 The COMPOSITION Architecture Specification I* and *D6.3 COMPOSITION Marketplace I*, as well as the communication protocols they use for interacting among each other. The agents belonging to the former category are in charge of providing crucial services for the marketplace to operate, while the latter is composed by the agents developed and deployed by the marketplace stakeholders. A Stakeholder agent could be a Requester agent, who can request the execution of an existing supply chain or to initiate a new one, or a Supplier agent, that is the related counterpart, able to respond to supply requests.

The usage of a reputation model will help the two classes of Stakeholder agents for making trust-based decision, for deciding which entities could be selected before initiating new transactions. Starting from the inferences made in D2.3, two approaches could be considered: (i) an agent-level personal ranking system and (ii) a central and collaborative agent ranking system. The COMPOSITION Reputation model will combine both for obtaining a more powerful and robust model, introducing the following two crucial concepts:

- **Local reputation**: associated to the first approach. It is the reputation given by a single Requester Agent R to another Supplier Agent S. Each Requester is in charge of computing and updating these local reputations.
- **Global reputation**: associated to the second approach. For global reputation, we mean the reputation of a Supplier S, aggregating the local reputations assigned to him by all the Requester, or a sub-set of them, who interacted with S during some past transactions.

The idea is to let each Requester compute his local reputations with respect to the all the Suppliers involved in previous transactions. These values, being computed internally without any data directly provided by other entities, will have a higher impact in the final trust-based decision of the Requester itself. Indeed, it is easier to trust ourselves, than others. For this reason, many times, the local reputation could be enough, especially if evaluated considering a high number of interactions between a Requester and a Supplier.

The global reputation will be used as an extra feature for improving the reliability of the final choice, if explicitly chosen by the Requester, considering also experiences of other agents. It will be calculated by a trusted third party, in charge of collecting all the local reputations, and send it, to a specific Requester. In COMPOSITION, this third party role will be played by the Matchmaker agent, a special Marketplace agent, in charge of providing specific support services to Stakeholder agents. Besides, it could also evaluate an offer received by the Requester from a Supplier, if explicitly asked by the former.

In D4.2, the requirements of the reputation model have been identified, numbered from R1 to R9, and this nomenclature will be also used in this document. A brief state of the art review about them has also been provided, highlighting the important difference between the concepts of trust and reputation. A high-level explanation of how the elicited requirements should be satisfied has been given, as well as some initial insights which will be used as starting points for defining the model. Specific sub-sections will be dedicated to local and global reputation evaluation (Section 4.1 and Section 4.2 respectively). Moreover, the crucial role of the Matchmaker agent will be explained, especially regarding its role for the global reputation evaluation (Section 4.3). Finally, a summary of how the designed model will fulfil the identified requirements will be given in Section 4.4.

## 4.1 Local Reputation Computation

A generic pair (Requester agent R, Supplier agent S) will be considered for the explanation. Every time an interaction occurs between R and S, the former should be able to rate the latter, depending on how the negotiation terminated, including the rating in the last message sent to S. Optionally, R could also ask the Matchmaker to compute the rating on its behalf.

A rating will be simply an integer belonging to the interval [1; 5], able to express different levels of satisfaction (“Not Satisfied”, “Partially satisfied”, “Satisfied”, “Very Satisfied”, “Completely Satisfied”), allowing the discrimination of agents’ behaviour during each negotiation. More information about the ratings has been provided in D4.2. The local reputation will be computed performing a proper aggregation of these values, as implemented in others common online marketplace scenarios (Resnick et al, 2002) (Sabater et al, 2001) (Gutowska et al, 2009), through an aggregator operator, for also representing the evolution of each Supplier’s

behaviour over time. In D4.2 a first selection has been made, leaving, for the final choice, the Weighted Mean (WM) and the Ordered Weighted Mean (OWA).

The concept of reputation lifetime played a crucial role for the final decision: it imposes to assign lower relevance (weights) to older ratings, when performing the aggregation, weighting each of them through a time function $f(t, t_i)$, where t indicates the instant when the local reputation has been computed/updated, while $t_i$ represents the time when the $i^{th}$ rating has been provided, associated to the $i^{th}$ transaction between R and S. This approach has also been followed in (Sabater et al, 2001) and (Gutowska et al, 2009), which have been considered as a basis for the design of the COMPOSITION Reputation Model.

Regarding the final choice of the aggregator operator, the WM allows weighting each rating basing on its reliability, while the OWA allows performing a permutation of the overall set of ratings, ordering them in a decreasing way and assigning the weights depending on this new order: however, this implies losing the association between a specific weight and the related rating (Llamazares, 2011). For the local reputation evaluation, ordering decreasingly the ratings is meaningless, especially if the relationship between the rating itself and its weight is lost, considering that the latter should be, in this case, a time function, strictly depending on the former.

Always in (Llamazares, 2011), other operators have been introduced, such as the Weighted OWA (WOWA) and the Hybrid WM (HWM), for trying to obtain the benefits of both the WM and the OWA with a unique aggregator. However, the authors expressed some doubts associated to their questionable behaviour, concluding that they do not represent a fully convincing choice. Considering the specific scenario of the projects and the previous deductions, the Weighted Mean has been selected as the final aggregator operator, for the local reputation computation. The generic formula can be seen in Equation 1.

$$\mathbf{WM} = \frac{\sum_1^N \boldsymbol{R_i} * \boldsymbol{P_i}}{\sum_{i=1}^N \boldsymbol{P_i}} \quad (1)$$

Where:
**N** is the overall set of considered ratings.
$\mathbf{R_i}$ is the $i^{th}$ rating related to the $i^{th}$ interaction between the Requester R and the Supplier S.
$\mathbf{P_i}$ is the $i^{th}$ weight associated to $R_i$. In this case each $P_i$ will be represented by a time function $f(t, t_i)$, as stated before, for assigning higher weights to more recent ratings.

The weighting vector $P= [P_1,\ldots,Pn]$ can be easily normalized dividing each weight for their sum. In this way, the denominator of Equation 1 will be equal to one, and the formula will be the same of the one shown in D4.2. The updated version of each normalized weight $W_i$, can be seen in Equation 2:

$$\boldsymbol{W_i} = \frac{\boldsymbol{P_i}}{\sum_{j=1}^N \boldsymbol{P_j}} = \frac{\boldsymbol{f(t, t_i)}}{\sum_{j=1}^N \boldsymbol{f(t, t_j)}} \quad (2)$$

Where $\sum_{i=1}^{N} W_i$ is equal to one.

Besides, given that the weighted vector has been normalized and each $R_i$ is an integer belonging to the interval [1, 5], it can be easily proved that the final local reputation will be a real number included in the same interval. After these deductions, in Equation 3 the updated formula for evaluating the local reputation is shown:

$$\boldsymbol{LR_{R \to S}} = \frac{\sum_{i=1}^N \boldsymbol{R_i} * \boldsymbol{W_i}}{\sum_{i=1}^N \boldsymbol{W_i}} = \sum_{i=1}^N \boldsymbol{R_i W_i} = \sum_{i=1}^N \boldsymbol{R_i} \frac{\boldsymbol{f(t, t_i)}}{\sum_{j=1}^N \boldsymbol{f(t, t_j)}} \quad (\mathbf{3})$$

Where $LR_{R \to S}$ is the local reputation assigned by Requester R to Supplier S.

As mentioned before, N represents the number of considered ratings, given by R to S. considering N past transactions. In (Sabater et al, 2001), the authors associated the growth of the opinions number (ratings number, in this case) to an increase of the reliability degree, until reaching a value they called “intimate level of interaction”. This value represents a close relationship from a social point of view, and could be considered as a sort of upper threshold, named $N_{max}$, for the total number of ratings used for local reputations computation. More ratings, and more transactions, will not increase the reliability degree, once that number has been

reached. This implies that the value of N, used in Equation 3, will be an integer belonging to the interval [1, $N_{max}$]. The value of $N_{max}$ is totally application dependent, and it could depend on many factors, such as the frequency of the interactions among the agents belonging to the marketplace, as well as the quality of ratings.

For concluding the section associated to local reputation, some experimental results will be shown, to infer what kind of time function could represents a good choice for the weights generation. Some insights about $N_{max}$ will also be provided.

### 4.1.1 Local Reputation Experimental Results

For the design of $f(t, t_i)$, five possible candidates have been considered, and named as follows:

1. **WM-REC**: $f(t, t_i) = {}^{1}/_{(t - t_i)}$
2. **WM-SQRT**: $f(t, t_i) = {}^{1}/_{\sqrt{t - t_i}}$
3. **WM-EXP**: $f(t, t_i) = {}^{1}/_{(t - t_i)^2}$
4. **WM-LOG**: $f(t, t_i) = {}^{1}/_{\ln(t - t_i)}$
5. **WM-REG**: $f(t, t_i) = {}^{t_i}/_{t}$

The last option has been included considering the REGRET model (Sabater et al, 2001), while the others represent common choices, for assigning higher weights to more recent ratings. Besides, being t always higher than $t_i$, the division by zero is avoided.

An experiment has been set up to infer what could be the best choice for this specific scenario. Moreover, also the Arithmetic Mean has been included in the experiment. The formula can be found in D4.2. In (Garcin et al, 2009) the authors expressed many negative opinions about using this operator for aggregating reputation feedback. Indeed, it has been included just for comparing its results with the ones obtained considering the other cases.

A new rating has been generated during each iteration cycle of the experiment, for a total of 200 iterations. A pseudo random generator has been used for selecting pseudo random ratings and timestamps, uniformly distributed in specified intervals, for having a good control of the overall simulation:

- **Iterations from 1 to 99**: low ratings have been chosen, belonging to the interval [1, 3], with each timestamp $t_i$ related to dates from January 1st 2018 from February 1st 2018.
- **Iterations from 100 to 149**: medium ratings have been selected, but this time the interval [2, 3] has been chosen. The timestamps $t_i$ belong to a more recent time interval, from March 1st 2018 from April 1st 2018
- **Iterations from 150 to 200**: the last ratings, instead, represent excellent transactions, in terms of satisfaction, indeed, the considered interval was [4, 5]. The time interval for the timestamps $t_i$ was the most recent, from May 1st 2018 from June 1st 2018

In Figure 10 the results of the experiment are represented.

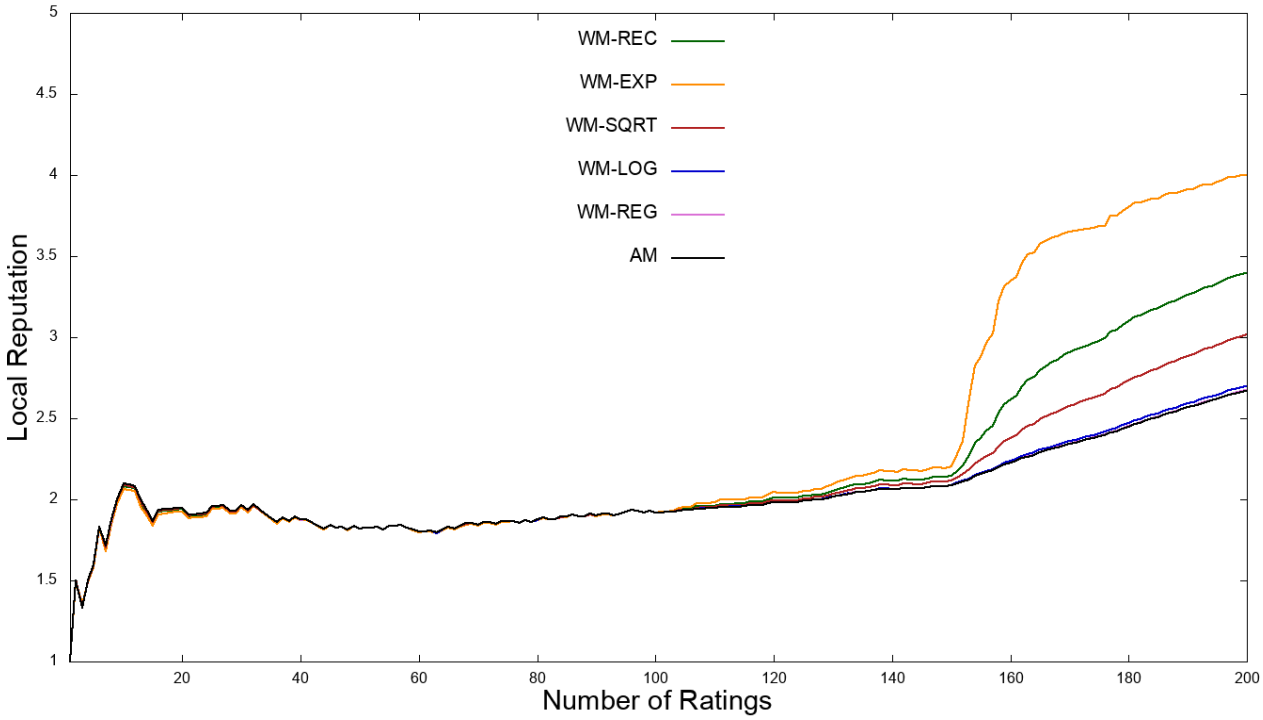


**Figure 10 Local Reputation First Simulation**

As expected, the Arithmetic Mean (AM) is not able to assign more relevance to more recent ratings. Similar results are obtained considering WM-REG and WM-LOG as time functions for evaluating weights, for this reason they were discarded. WM-SQRT is better from this point of view, but there is still not a great difference. Instead, interesting inferences can be deducted from the other cases. More precisely, the WM-EXP function gives high importance to newer ratings. However, in a generic online marketplace scenario, some misleading ratings could be provided, for underestimating, or overestimating, a reputation value. Some agents could choose to provide a fake rating, for example when interacting with a competitor, for decreasing his reputation. Or simply, they could perform some incorrect evaluation, providing a wrong rating.

Using WM-EXP as time function could potentially help this behaviour: malicious agents could provide some recent fake ratings, influencing negatively the associated local reputation of the victim. For this reason, a good choice could be represented by a trade-off between the behaviour of WM-EXP and WM-SQRT. From this point of view, as can be seen in Figure 10, WM-REC could satisfy this requested trade-off. This potential problem could be solved by the Matchmaker agent, as it will be explained more in detail in Section 4.3.

A second experiment has also been set up, always considering 200 iterations, characterised by the following parameters, where timestamps have been selected considering the same criteria of the previous case:

- **Iterations from 1 to 99**: medium ratings have been chosen, belonging to the interval [2, 3].
- **Iterations from 100 to 149**: good ratings have been selected, but this time the interval [3, 4] has been chosen.
- **Iterations from 150 to 200**: very bad ratings have been considered, included in the interval [1, 2].

The results are shown in Figure 11.

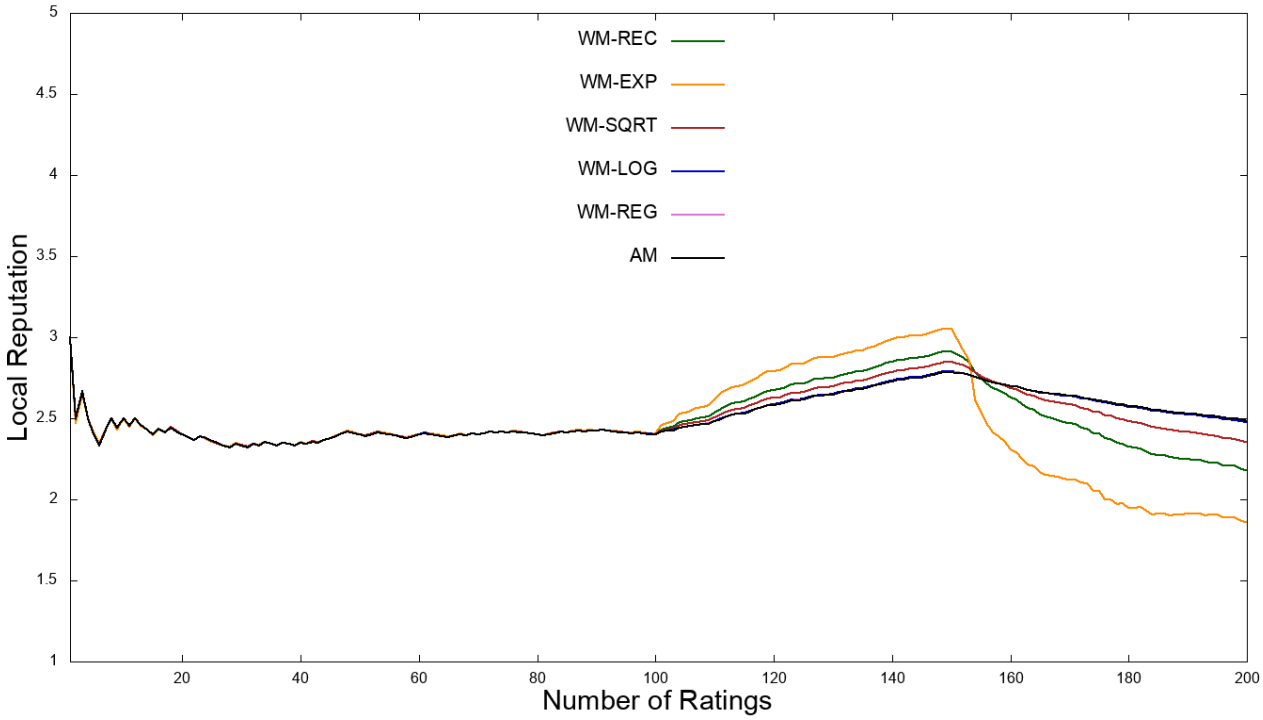


**Figure 11 Local Reputation Second Simulation**

As expected, the WM-REC function allows obtaining the needed trade-off between WM-SQRT and WM-EXP, also when higher weights should be assigned to bad ratings.

Before concluding, it is important to highlight that these experiments will never represent a real scenario, especially the last one. Indeed, it is practically impossible that a Requester will continue to interact with a Supplier after 50 unsatisfied transactions. We just used them for inferring a good time function to be used for weights generation, which fits well for the COMPOSITION Marketplace scenario. The objective was to give more importance to more recent transactions, trying, at the same time, to minimize the impact of possible false rating. Indeed, the WM-REC time function has been chosen. The final formula of the local reputation is represented in Equation 4.

$$\boldsymbol{LR_{R \to S}} = \sum_{i=1}^{N} \boldsymbol{R_i} \frac{1/(t - t_i)}{\sum_{j=1}^{N} 1/(t - t_j)} \quad (\mathbf{4})$$

Regarding the final choice of $N_{max}$, more experiments will be set up and the results presented in the next deliverable, when the prototypes of the COMPOSITION Agents will be also available.

As mentioned before, local reputations are computed and maintained locally by each Requester, who should keep all the ratings it needs for this computation. In the experiments, 200 ratings have been considered, but a less number will also be sufficient for providing reliable results. An idea could be to aggregate very old ratings, through the same formula represented in Equation 4 and converting the result to an integer, using it as a single rating for the overall computation, once the number of interactions increases. In this way, the local reputation will still represent the historical behaviour of a Supplier, but limiting the total number of ratings that must be locally maintained.

Moreover, Requesters are in charge of sending each updated value to the Matchmaker agent, for letting him able to compute global reputations, when needed. Besides, local reputations must be continuously checked, not only by the Matchmaker, for detecting any possible incorrectness, considering that they could be used, indirectly, by other Requesters. The COMPOSITION Blockchain, described in D2.3, could be exploited, for this purpose. More in detail, it does not allow the storage of the value itself, but it provides the functionality to store its hash. Indeed, every time a new local reputation is computed by a Requester, the related hash will be stored in the Blockchain. Obviously, only Requesters are authorized to store and modify hashes of local reputations they directly computed, with respect to specific Suppliers. A Supplier should not be authorized to do that, as well as a Requester on behalf of another Requester.

Regarding the checking operation, the Supplier is aware of all the ratings assigned to him by the Requester itself during all their transactions. He can aggregate them and check the correctness of the reputation, thanks to the functionalities offered by the COMPOSITION Blockchain and the peculiarity of the hash functions. If a mismatch is obtained, then the Supplier could warn the Matchmaker, who, in turn, will perform the final checking, requesting the needed information to both the Requester and the Supplier, if necessary. This methodology will allow also inferring if all the agents are using the same aggregator operator for the calculation of the local reputation.

Section 4.2 will be dedicated to global reputation, and it will follow the same structure of Section 4.1.

## 4.2 Global Reputation Computation

Global reputation has been defined at the beginning of Section 4, and it could be the result of a central and collaborative ranking system, able to compute the reputation of a generic Supplier S, considering the local reputations given by all the Requesters, or a sub-set of them, which interacted with S during some past transactions. It is computed by the Matchmaker agent, for this reason, being a common trusted third party, both Requester and Supplier should not be worried about possible malicious behaviours.

However, as already explained in D4.2, when considering different local reputations given by different agents, their credibility must be taken into account: global reputation is calculated on behalf on a specific Requester, and his point of view must be taken into account, when experiences of other agents are aggregated. In (Gutowska et al, 2009) this factor was called rater's credibility.

Considering the usual pair (Requester R, Supplier S), the usefulness of global reputation is explained through the possibility to let R relying also on other agent's opinions, and not only on its local reputations, before starting a new interaction with S, for making a more reliable trust-based decision. Besides, global reputations can also be very helpful when R and S should interact for the first time: in this case, indeed, R does not have any local reputation associated to S, but he could get some hints from its global reputation, about its behaviour with other agents.

For dealing with the problem of rater's credibility, the global reputation of S will not be the same for each Requester agent. Indeed, local reputations evaluated by other agents will be weighted properly, according to the specific Requester who needs the global reputation. For instance, considering the point of view of R, the idea is to (i) select a sub-set of R's trusted Requesters, which interacted with S in the past, and (ii) weight their opinions towards S with the local reputations assigned by R to them, for obtaining a more reliable result.

Regarding the choice of the aggregator operator, there are not any significant differences with respect to the previous case. Indeed, also for global reputation, there is a direct connection between the weight and the value to be weighted, and this connection must be maintained: for this reason, the same conclusions made in Section 4.1 are still valid. The WM aggregator will represent the final choice, also for this scenario. The generic formula is represented in Equation 5.

$$\boldsymbol{GR_{R\rightarrow S}} = \frac{\sum_{\boldsymbol{Ri\in Ir}} \boldsymbol{LR_{Ri\rightarrow S} f(LR_{R\rightarrow Ri})}}{\sum_{\boldsymbol{Ri\in Ir}} \boldsymbol{f(LR_{R\rightarrow Ri})}} \quad (\mathbf{5})$$

Where:
$\mathbf{GR_{R\rightarrow S}}$ is the global reputation of S, evaluated with respect to R.
$\mathbf{I_R}$ is the trusted sub-set of considered Requesters for the computation of the global reputation
$\mathbf{R_i}$ is the $i^{th}$ Requester belonging to $I_R$
$\mathbf{LR_{Ri\rightarrow S}}$ is the local reputation given by $R_i$ to S
$\mathbf{f(LR_{R\rightarrow Ri})}$ is a function of the local reputation given by R to $R_i$, who acts as a weight

For normalizing the weights, the same approach used for the local reputation has been followed. The $i^{th}$ normalized weight is represented through Equation 6:

$$\boldsymbol{W_i} = \frac{\boldsymbol{f(LR_{R\rightarrow Ri})}}{\sum_{\boldsymbol{Rj\in Ir}} \boldsymbol{f(LR_{R\rightarrow Rj})}} \quad (\mathbf{6})$$

Where $\sum_{\mathrm{Ri\in Ir}} W_i$ is equal to one.

Besides, given that the weighted vector has been normalized and each $\mathrm{LR_{Ri\rightarrow S}}$ is a real number belonging to the interval [1, 5], also the final global reputation will be a real number included in the same interval. The updated formula can be seen in Equation 7:

$$\boldsymbol{GR_{R\rightarrow S}} = \frac{\sum_{\boldsymbol{Ri\in Ir}} \boldsymbol{LR_{Ri\rightarrow S} W_i}}{\sum_{\boldsymbol{Ri\in Ir}} \boldsymbol{W_i}} = \sum_{\boldsymbol{Ri\in Ir}} \boldsymbol{LR_{Ri\rightarrow S} W_i} = \sum_{\boldsymbol{Ri\in Ir}} \boldsymbol{LR_{Ri\rightarrow S}} \frac{\boldsymbol{f(LR_{R\rightarrow Ri})}}{\sum_{\boldsymbol{Rj\in Ir}} \boldsymbol{f(LR_{R\rightarrow Rj})}} \quad (7)$$

However, at this point, a clarification is needed, before proceeding with the following sections. A Requester can rate only a Supplier. For this reason, saying that the local reputation calculated by R with respect to other Requesters will be used is not properly correct. As stated in D2.3, "factories transforming goods typically employ at least one Requester agent, to get prime goods and one supplier agent to sell intermediate products to other factories". This means that the local reputation evaluated by R is not related to $R_i$, but to its counterpart $S_i$. The same deduction is also valid for the $I_R$ members: indeed, the choice will be made considered a trusted set of pair (Requester, Supplier) associated to the same stakeholder.

The updated version of the formula is shown in Equation 8:

$$\boldsymbol{GR_{R\rightarrow S}} = \sum_{\boldsymbol{(Ri,Si)\in Ir}} \boldsymbol{LR_{Ri\rightarrow S}} \frac{\boldsymbol{f(LR_{R\rightarrow Si})}}{\sum_{\boldsymbol{(Rj,Sj)\in Ir}} \boldsymbol{f(LR_{R\rightarrow Sj})}} \quad (8)$$

If there are no pairs which satisfy the criterion for belonging to $I_R$, then the Matchmaker could use the Arithmetic Mean for evaluating the global reputation, selecting a group of Requesters who rated S in the past, without weighting their experiences. The value will not have a high reliability, but it could represent a good hint for the Requester who needs it. Anyway, this will not be a huge problem because, in this case, the Requester will often consider just the local reputations for making the final choice.

The worst-case scenario is characterised by the total absence of R's local reputations, for example when he just joined the marketplace. In this case, exploiting the global reputation evaluated with a simple Arithmetic Mean represents a simple way for having, at least, a hint about the historical behaviour of a Supplier, evaluated by a trusted third party, aggregating experiences of different agents. This will allow the new member of the marketplace starting the interactions with the other agents and build its own local reputations.

As for the local reputation, some experimental results will be shown in Section 4.2.1, aiming at defining the appropriate $\mathrm{f(LR_{R\rightarrow Ri})}$ to be used in the evaluation. Besides, the criteria for the selection of the Requester belonging to $I_R$ will be identified.

### 4.2.1 Global Reputation Experimental Results

Being aware of all the local reputations evaluated by each Requester, with respect to each Supplier, the Matchmaker can easily generate a trusted group for a specific Requester R, simply considering the local reputation values. These values must be higher than, or equals to, a specific threshold $T_R$.

Considering the different levels of satisfaction listed in D4.2, a good choice for $T_R$ could be represented by an intermediate value between the "Partially Satisfied" and the "Satisfied" levels. From a numeric perspective, the chosen value could be 2.5. This implies that all the Suppliers, which have a local reputation, evaluated by R, higher than, or equal to, 2.5, and an associated Requester agent, can be part of $I_R$.

Regarding the design of $\mathrm{f(LR_{R\rightarrow Ri})}$, two candidates have been considered, and named as follows:

- **WM-MUL:** $f(LR_{R\rightarrow Ri}) = \mathrm{LR_{R\rightarrow Si}}$
- **WM-E:** $f(LR_{R\rightarrow Ri}) = \mathrm{e^{LR_{R\rightarrow Si}}}$

As in Section 4.1.1, the Arithmetic Mean will be also considered, for the same motivation.

For the experiments, during each cycle a new member has been added in $I_R$, for a total of 50 iterations. As for the previous simulations, a pseudo random generator has been used for selecting pseudo random local reputations, uniformly distributed in specified intervals. In the first case, the following parameters have been considered:

- **Iterations from 1 to 24**: good values for both $\mathrm{LR_{Ri\rightarrow S}}$**,** belonging to the interval [3, 4], and $\mathrm{LR_{R\rightarrow Si}}$, included in the interval [3, 3.5].

- **Iterations from 25 to 37**: medium values for both $LR_{Ri \rightarrow S}$, belonging to the interval [2, 3], and $LR_{R \rightarrow Si}$, included in the interval [2.5, 3].
- **Iterations from 38 to 50**: very low values for $LR_{Ri \rightarrow S}$, belonging to the interval [1, 2], while high values have been considered for $LR_{R \rightarrow Si}$, included in the interval [4.5, 5].

The results of the experiment have shown in Figure 12.

As expected, there are no big differences between WM-MUL and the Arithmetic Mean. The former simply uses each $LR_{R \rightarrow Si}$ as weights, belonging to the interval [$T_R$, 5], considering the selection rule for $I_R$. The interval length is quite short, and using this kind of function does not allow weighting sufficiently each $LR_{Ri \rightarrow S}$. Better results, instead, are obtained with the WM-E function: in this case, exploiting the exponential function allows assigning higher weights for higher values of $LR_{R \rightarrow Si}$, as can be seen in Figure 12, focusing on the last 12 iterations.

In the second experiments, the used parameters are listed below:

- **Iterations from 1 to 24**: medium values for $LR_{Ri \rightarrow S}$, belonging to the interval [2, 3], and quite good values for $LR_{R \rightarrow Si}$, included in the interval [3, 3.5].
- **Iterations from 25 to 37**: quite good values for $LR_{Ri \rightarrow S}$, belonging to the interval [3, 3.5], and medium values for $LR_{R \rightarrow Si}$, included in the interval [2.5, 3].
- **Iterations from 38 to 50**: very high values for both $LR_{Ri \rightarrow S}$, belonging to the interval [4,5, 5], and $LR_{R \rightarrow Si}$, also included in the interval [4.5, 5].

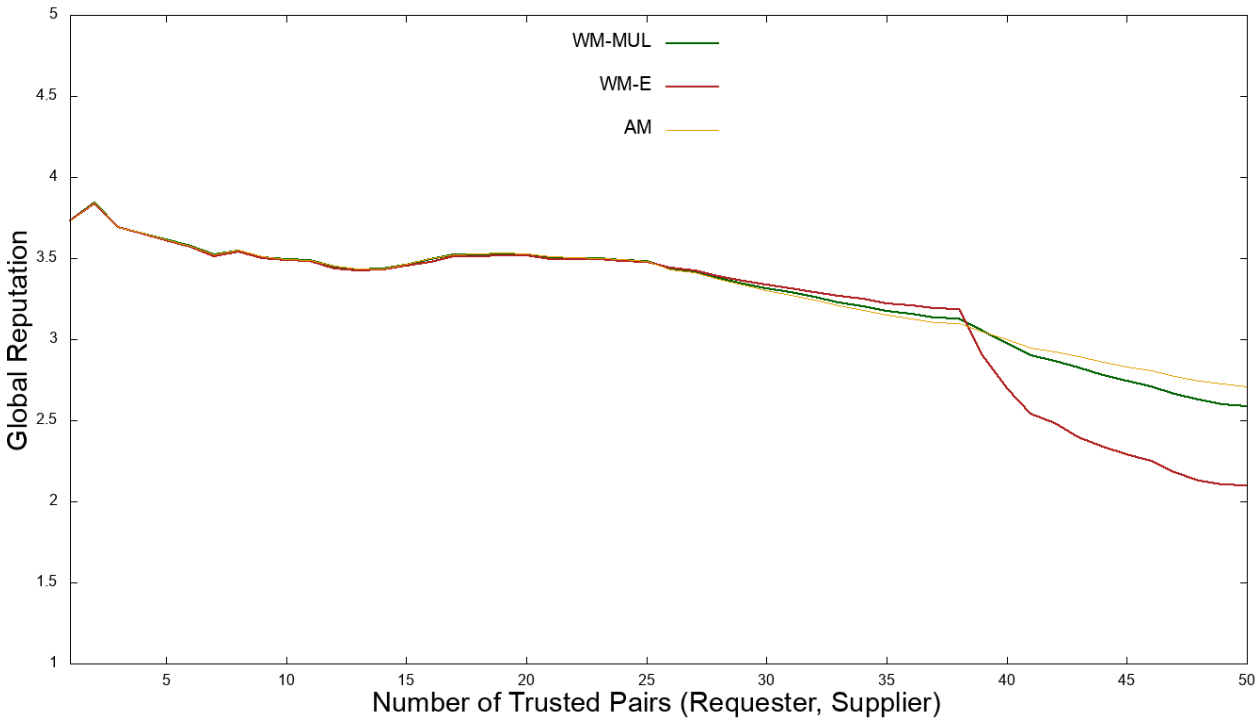


**Figure 12 Global Reputation First Simulation**

The results of the second experiment are shown in Figure 13.

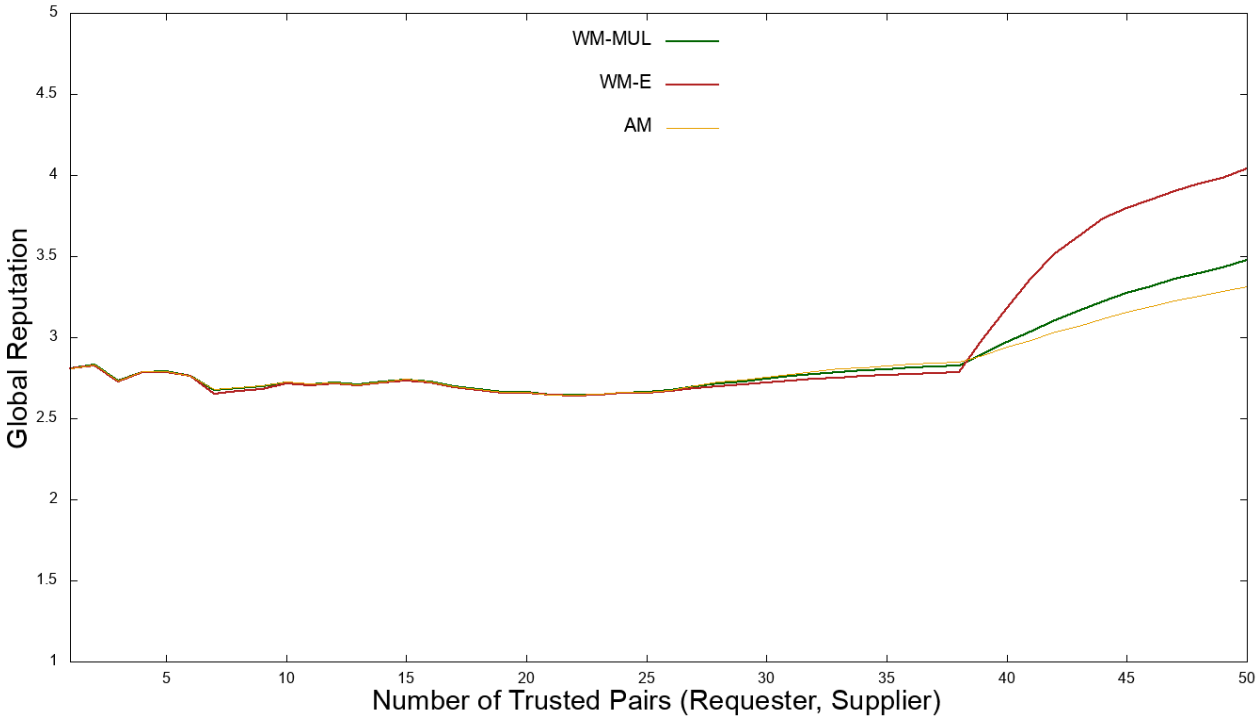


**Figure 13 Global Reputation Second Simulation**

As expected, WM-E function correctly assigns a high relevance to the last 12 local reputation values, with respect to WM-MUL and the Arithmetic Mean. The complete formula of the global reputation is shown in Equation 9:

$$GR_{R \to S} = \sum_{(Ri,Si) \in Ir} LR_{Ri \to S} \frac{e^{LR_{R \to Si}}}{\sum_{(Rj,Sj) \in Ir} e^{LR_{R \to Sj}}} \quad (9)$$

The next section will be dedicated to the role of the Matchmaker agent, from the point of view of the COMPOSITION Reputation Model.

## 4.3 Role of the Matchmaker Agent

The Matchmaker agent is a Marketplace agent belonging to the COMPOSITION environment. Very briefly, it offers a set of REST APIs for providing specific services to Stakeholders agents. Its features ad roles are described in detail in D2.3 and D6.3. More precisely, it is also the agent who provides the set of Suppliers who could be contacted by the Requester R for initiating a negotiation.

Considering the COMPOSITION Reputation Model, it is in charge of the following operations:

1. **Global Reputation computation**: maybe, the more important task. After computing/updating a local reputation, the Requester sends the value to the Matchmaker. In this way, the latter, possesses all the local reputations evaluated by each Requesters, becoming able to compute accordingly global reputations on behalf of them. This will have a positive impact also for the selection of the set of Supplier that must be returned to the involved Requester, before initiating a new transaction, considering that the Matchmaker could rely on more information for inferring this final decision.
2. **Provide ratings for single transactions between a Requester and a Supplier**: the Matchmaker could rate the Supplier after a transaction initiated by a specific Requester, if explicitly asked by the latter. If all the ratings would be provided by the Matchmaker, the problem of fake ratings will be totally solved, considering that he acts as a totally trusted third party.

3. **Local reputations check**: as mentioned before, a Requester is in charge of storing each updated local reputation in the COMPOSITION Blockchain, while the Supplier has the possibility to check it. If a mismatch is obtained, the Matchmaker could act as a "referee", for determining who is trying to cheat, or simply who performed the incorrect evaluation. If he possesses all the needed data (ratings and timestamps) this could be done automatically, otherwise both the Requester and the Supplier should provide the missing information to him.
4. **Provide the set of possible Supplier to be contacted by the involved Requester**: before initiating a transaction, a Requester contacts the Matchmaker, who responds with a set of possible Supplier that can be contacted. Then, the Requester could use the local reputations and, optionally, the global reputations evaluated by the Matchmaker itself, for making the final decision.

Scalability issues could impact the overall performance. However, they have already been discussed in detail in D2.3 and D6.3, and the same analysis will not be repeated in this document.

## 4.4 Requirements Fulfilment

This section will just provide a final summary, detailing how the designed model is able to fulfil the requirements stated in D4.2.

**R1 - Reputation and ratings should discriminate agents' behaviour.** A rating will be simply an integer belonging to the interval [1; 5], able to express different levels of satisfaction ("Not Satisfied", "Partially satisfied", "Satisfied", "Very Satisfied", "Completely Satisfied"), allowing the discrimination of agents' behaviour during each negotiation. Local reputations, in turn, are evaluated aggregated ratings associated newer and older transactions, obtaining a representation of the agents' behaviour over time.

**R2 - Incorrect reputation values should be detected (rater's credibility), when used.** Incorrect local reputations are detected thanks to the usage of the COMPOSITION Blockchain, as explained in Section 4.1.1 and Section 4.3. Global reputations do not have this problem, considering that they are directly evaluated by a trusted third party. The only potential problem is related to fake ratings, as explained in Section 4.1.1. However, this has been minimized by the choice of a proper weight function for the calculation of the local reputation. Moreover, for totally solve this potential problem, each rating could be computed directly by the Matchmaker.

**R3 - Local reputations should be available to all the agents belonging to the marketplace, if needed.** This requirement was stated before considering the potential role of the Matchmaker, regarding the global reputation computation, introducing this centralized entity. He is the only one who needs these values, indeed each Requester must provide them to him, after each computation/update. However, each agent will have the possibility to ask directly to the Matchmaker for some specific local reputation values evaluated by other Requester, if needed.

**R4 - Reputation lifetime must be taken into account.** The usage of a specific time function, as weight function for the computation of local reputations, allowed assigning more relevance to more recent ratings, as requested explicitly by the context of reputation lifetime.

**R5 - Agents should not be able to compute, or modify, their own reputation value.** Only Requesters are authorized to store, and eventually modify, the hash of the local reputations they compute, associated to other Suppliers, in the COMPOSITION Blockchain. However, considering the peculiarity of the COMPOSITION Marketplace, Suppliers can evaluate their own reputation, given to them by a specific Requester, just in order to check the correctness of the stored value. If a mismatch occurs, the Matchmaker will make the final decision, eventually forcing the Requester to modify the value previously stored. This change has been considered critical for performing this integrity check.

**R6 - Involved agents must use the same aggregator operator.** Using the COMPOSITION Blockchain, allowing Suppliers and, eventually the Matchmaker, to check the integrity of a local reputation, automatically fulfil this requirement, as explained in Section 4.2.1. For the global reputation, there are not any problems, considering that only the Matchmaker will compute these values.

**R7 - Reputation values must represent the evolution of the agent's behaviour.** The same explanation of R2 could be used for explaining how this requirement has been fulfilled.

**R8 - New agents should not be penalized.** Considering that the choice of the sub-set of returned agents to be contacted will be made by the Matchmaker, a specific policy can be implemented in order to not penalize

new agents who recently joined the marketplace. For example, default global reputations could be assigned to them**,** to allow other entities initiating transactions with them. This default value should not be too low, and also not too high, for not penalizing older agents who built their good reputation over time. A default value of 2.5 could represent a good choice, considered the levels of satisfactions associated to ratings/reputations.

**R9 - "Bad" agents should not be able to leave the marketplace and re-join as different agents.** This was the only requirement not explicitly considered in the previous sections. Fortunately, the characteristics of the COMPOSITION Marketplace allow fulfilling it in an easy way. In COMPOSITION, there is a special Marketplace agent, called White Pages agent. The agent is in charge of tracing all the agents operating on the marketplace, providing information about their status, as described in D2.3. Indeed, it is practically impossible for a known agent to leave the marketplace and re-join again, as a different agent, for deleting the history associated to its bad reputation. It could just become "inactive", but, in this case, its reputations will be maintained, and re-used once the status changes again to "active".

This summary concludes the section about the design of the COMPOSITION Reputation Model. Future work will be related to how this model could be implemented by the agents belonging to the marketplace, exploiting the COMPOSITION Blockchain, as explained in the previous sections. Once the first prototypes, of both the agents and the Blockchain, will be available, some specific tests could be performed, and the results presented in *D4.5 Prototype of the Security Framework II*.

# 5 Deployment

For this first version of the Security Framework prototype, the components have been deployed in ATOS servers due to avoid connectivity problems and ensure that all the external components are secured in a proper way. Here, the deployment of the most representative components is described. Other technical components, such as MySQL… have been deployed by different partner's IT teams, but its technical description is out of the scope of this document.

The architecture used as guidelines for the deployment is described in detail in *D4.1 Design of the Security Framework I* and *D4.2 Design of the Security Framework II*:

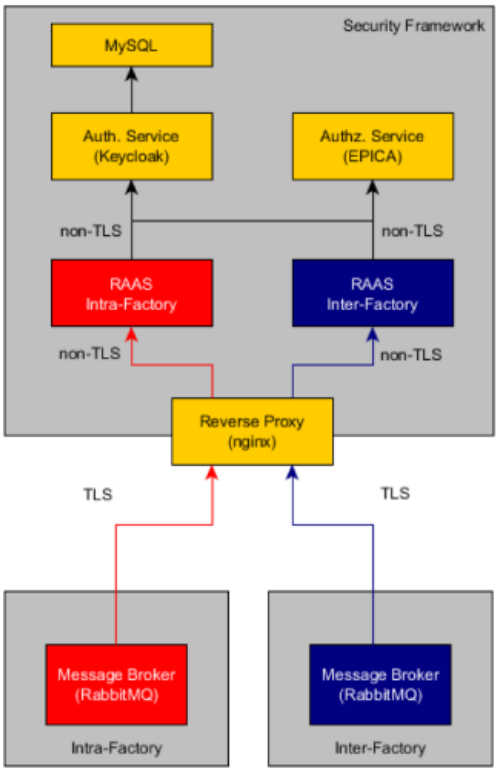


**Figure 14 Security Framework architecture**

## 5.1 EPICA Deployment

EPICA has been deployed using Docker[13] in Atos premises. More precisely, two docker containers have been created, one for the PAP and the other for the Authorization Engine, called paapi and webpep respectively, as can be seen in the screenshot in Figure 15, taken from Portainer dashboard[14].

[13] https://www.docker.com/
[14] https://portainer.io/

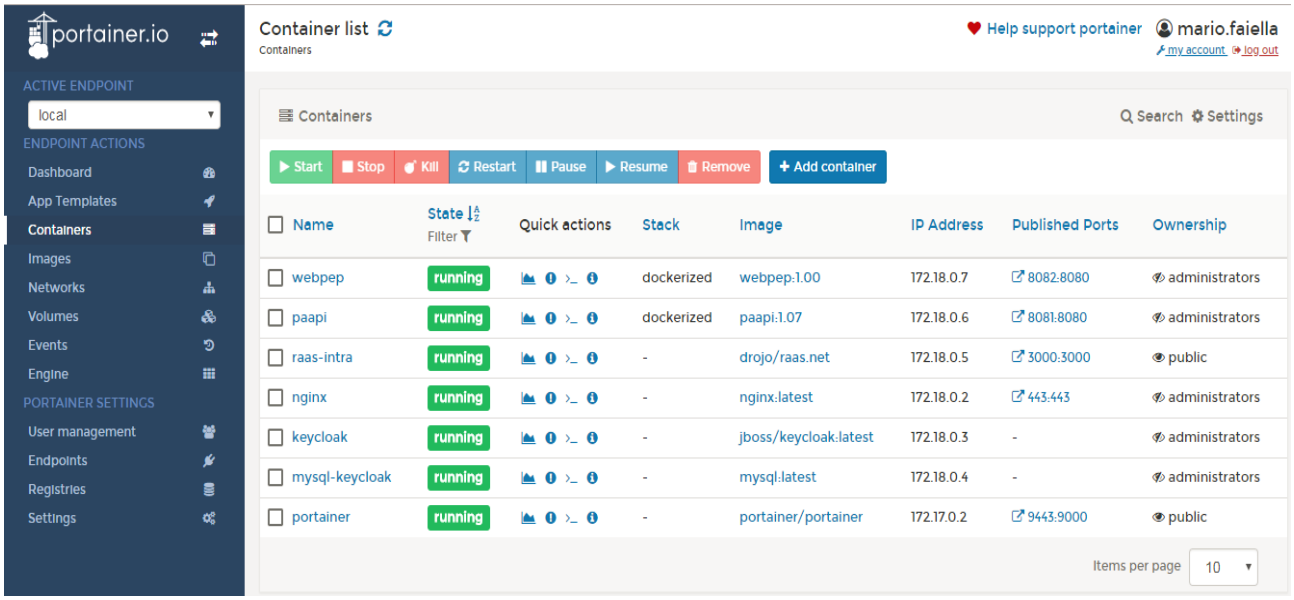


**Figure 15 Portainer Dashboard**

For their definition and deployment, the Docker Compose tool[15] has been used, which is widely adopted when dealing with multi-container Docker applications. In this way, it is possible for the containers belonging to EPICA to share the same private internal network, making easier the interaction among them, especially if one or more PIPs should be added in the future, for improving the policy enforcement process, increasing the overall number of containers associated to EPICA.

All the components of the Security Framework must share the same private internal network. For this reason, these two containers have been added to the already existing “security-framework” network. Its details can be seen in Figure 16.

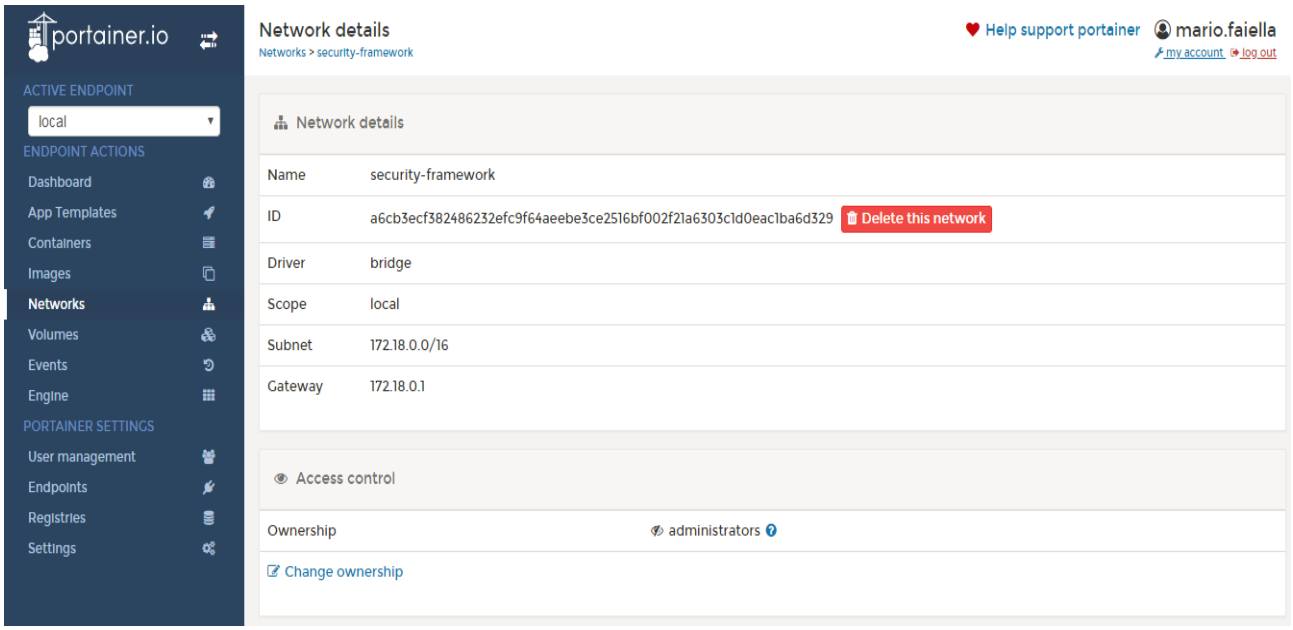


**Figure 16 Security Framework Private Network**

Moreover, two Docker Volumes[16] have been defined for persisting data generated by the involved containers. The content of the volumes is independent from the container’s lifecycle, making it possible to check stored

[15] https://docs.docker.com/compose/
[16] https://docs.docker.com/storage/volumes/

information even after removing or shutting down the containers. For the usage of EPICA in COMPOSITION, this is interesting for two different purposes: (i) policies storage and (ii) logs storage. In the former case, policies could be stored permanently in the local memory, while in the latter, instead, logs related to policy enforcement process and policy management could be consulted anytime. The details of the volumes can be seen in Figure 17 and Figure 18 respectively. New volumes could be added if needed, especially if new containers will be needed.

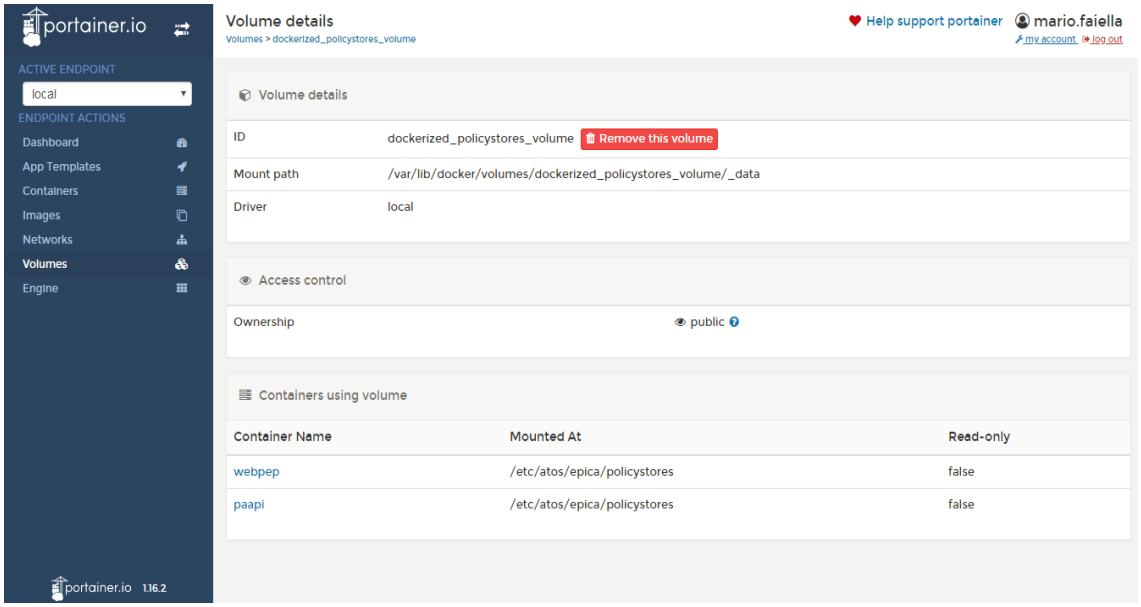


**Figure 17 Policy Storage Volume**

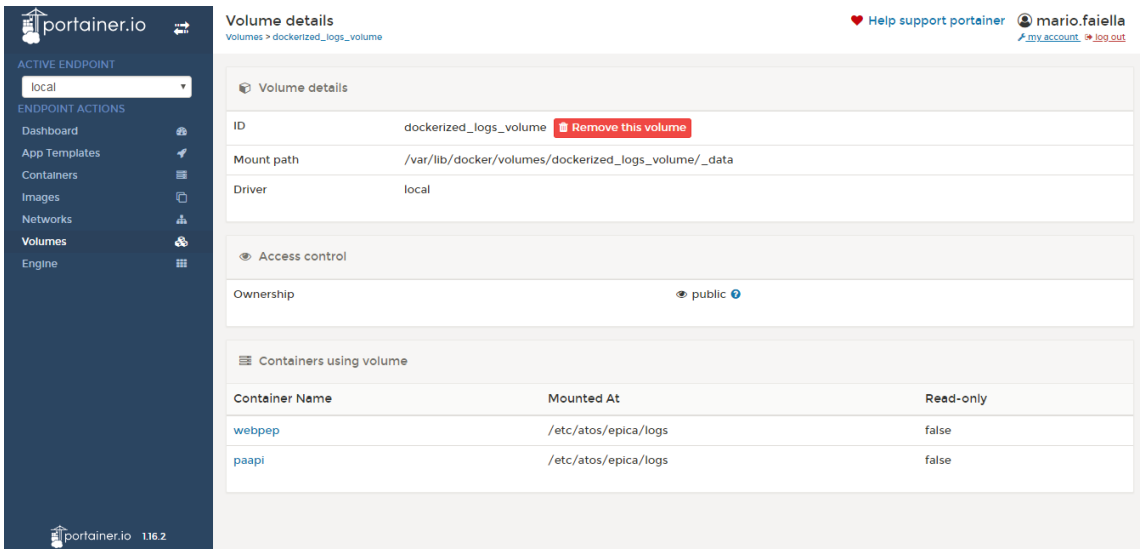


**Figure 18 Log Storage Volume**

## 5.2 Keycloak

The election of Keycloak as single sign on solution for the securing of the Security Framework is due to its simplicity. In this section, we'll describe how to implement security features in different environments as the instances of inter-factory and intra-factory we have deployed in the COMPOSITION project.

For further operations that are not described in this deliverable, go to Keycloak official documentation[17].

### 5.2.1 Admin console

The admin console is the core component of the management of the authentication service. The way to access is via URL, to the ATOS' server in which is deployed:

https://5.79.93.79/auth/

So we enter to the welcome page:


**Figure 19 Keycloak welcome page**

So, clicking in the "Administration Console" link we proceed to enter our credentials in the login page:


**Figure 20 Login page**

Once we enter the admin user and password, it'll bring us to the Keycloak Admin Console:

[17] https://www.keycloak.org/documentation.html

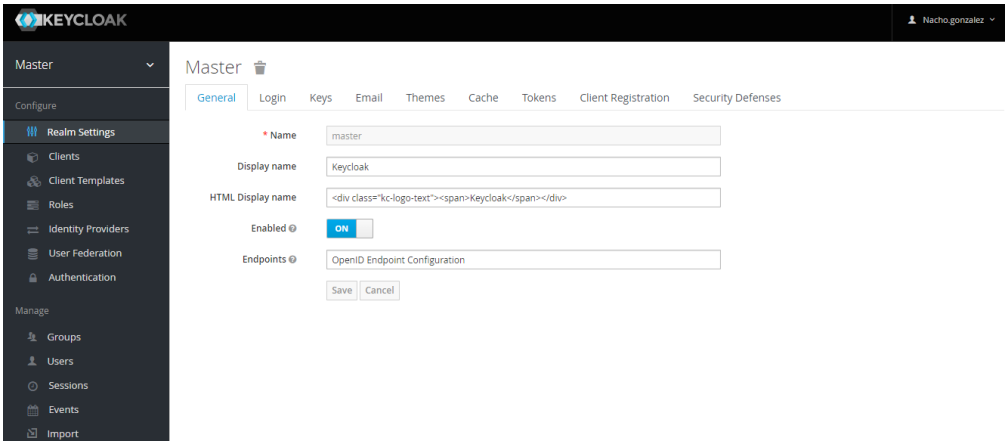


**Figure 21 Keycloak admin console**

The structure of this page is quite simple but functional. There is one menu on the left, where we can manage realms (by default it manages “Master” realm).

The menu on the right, just dropping off from the user name allows us to view our user account or logout.

Furthermore, there is a help system embedded. If we’re not sure about any certain field, we can hover our mouse over any question mark icon, and a tooltip text will appear providing more information about this field.

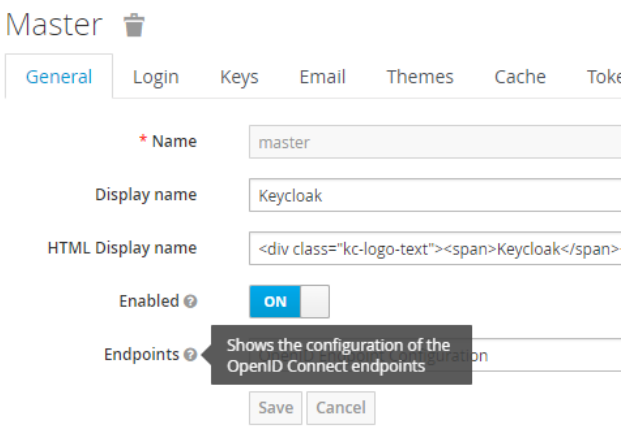


**Figure 22 Tooltip text in action**

#### 5.2.1.1 The Master realm

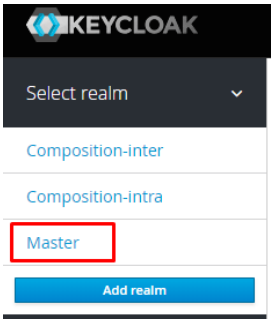


**Figure 23 The Master realm**

The Master realm is the default realm that Keycloak creates when is booted for the first time. Is in the highest level in the hierarchy tree of the realms, and the admin accounts enclosed in this realm have permissions to view and manage any other realm created on the server.

The use of the Master realm is recommended for super admins for creation and management of other business realms in the system. This security model prevents accidental changes and follows the tradition of permitting user accounts to only those privileges and powers necessary for the successful completion of their current task.

#### 5.2.1.2 Creating a new realm

For creating a new real, we've to click in the "Add realm" button that appears in the real drop-down menu on the left:

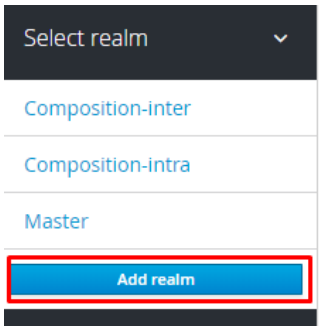


**Figure 24 "Add realm" button**

Then we complete the information in the Add Realm page. The information we have to provide here is only the realm name. In other way, we can import a JSON document that defines the new realm.

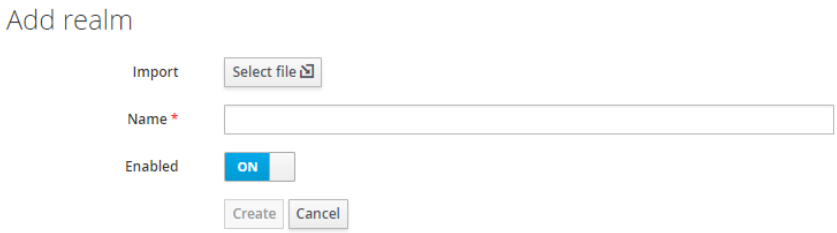


**Figure 25 "Add realm" page**

#### 5.2.1.3 SSL Mode

The SSL Mode defines the SSL/HTTPS requirements for interacting with the realm. If any application, browser or component doesn't honor the SSL/HTTPS requirements, it will not be allowed to interact with the server.

The configuration of the SSL Model of a realm is done in the Realm Settings page, clicking in the option of the menu:

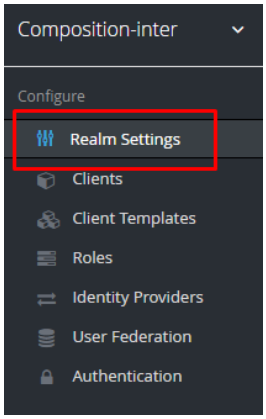


**Figure 26 Realm settings menu option**

In the Login tab, we have the "Require SSL" menu, where we'll select our SSL Mode:

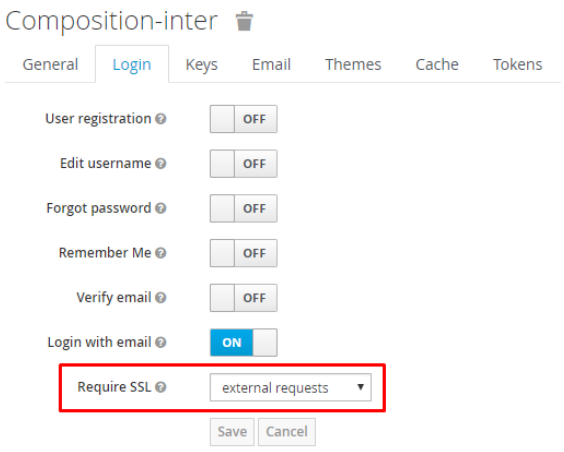


**Figure 27 Require SSL dropbox menu**

We have different SSL modes:

- **External requests**: Users can interact with Keycloak so long as they stick to private IP addresses like localhost, 127.0.0.1, 10.0.x.x, 192.168.x.x, and 172.16.x.x. If you try to access Keycloak from a non-private IP address you will get an error.
- **None**: Keycloak does not require SSL. This should really only be used in development when you are playing around with things and don't want to bother configuring SSL on your server
- **All requests**: Keycloak requires SSL for all IP addresses

#### 5.2.1.4 Clearing server caches

A very useful option for administrators is cleaning up the server caches. We can clear the realm cache, the user cache or external public keys from the Admin Console in the Realm Settings left menu item and the Cache tab.

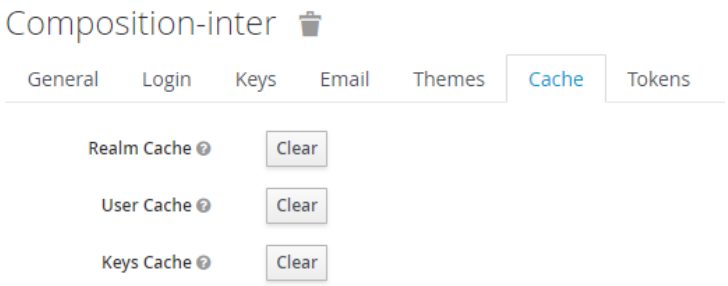


**Figure 28 Clear cache options**

### 5.2.2 User management

#### 5.2.2.1 User search

For searching a specific user, we just have to click in the “Users” menu option. A “Lookup” tab will appear and we can introduce the search information in the textbox:

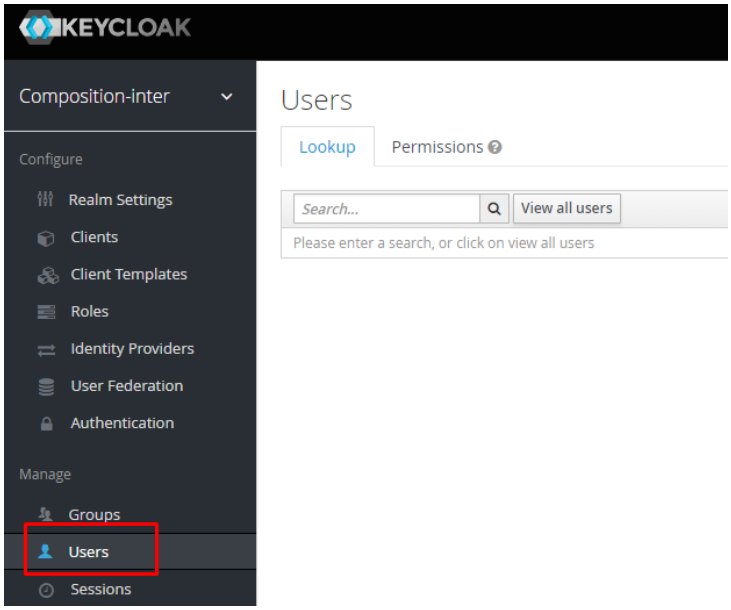


**Figure 29 Users search**

#### 5.2.2.2 Create new user

In the “Users” page (same page for the search user option), we can click on the “Add user” button on the right for creating a new user:

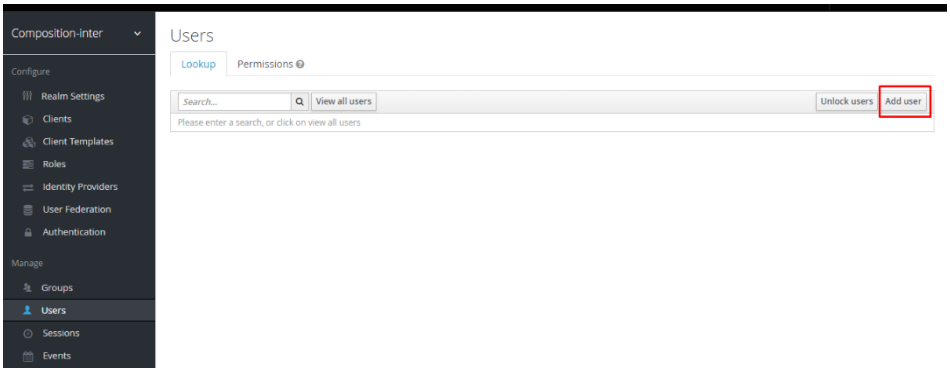


**Figure 30 Add user button**

So, we enter the page for entering the information of the new user:

- ID
- Username (only field required)
- Email
- First name
- Last name
- User enabled
- Email verified
- Required user actions

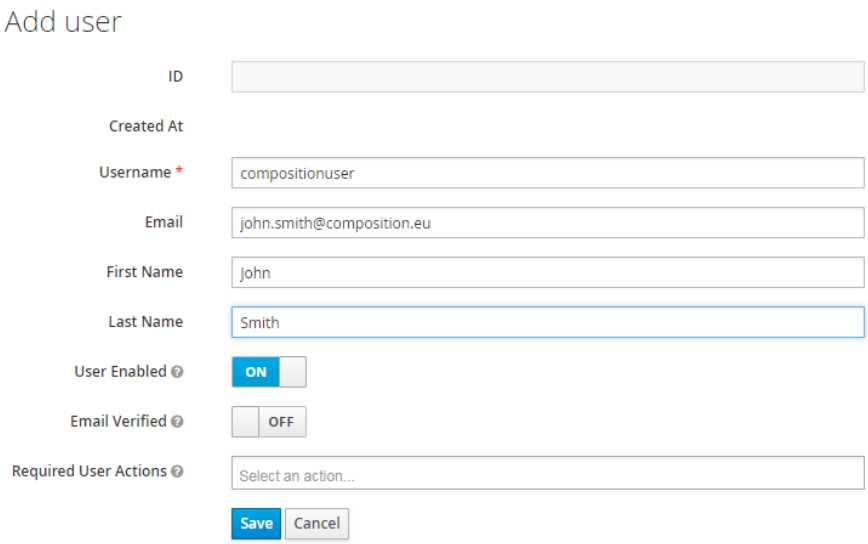


**Figure 31 Add user information**

### 5.2.2.3 Deleting users

For deleting one user, we'll have to search first as described in Section 5.2.2.1 the user we want to delete. For example, we're going to delete the user with username "compositionuser". Once we find the user, we can click on the "Delete" button on the right side of the row:

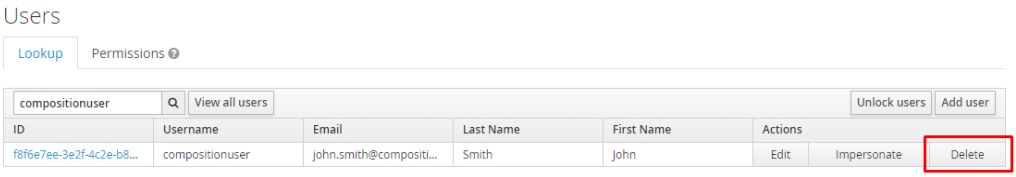


**Figure 32 Delete user button**

We'll be asked for our confirmation:

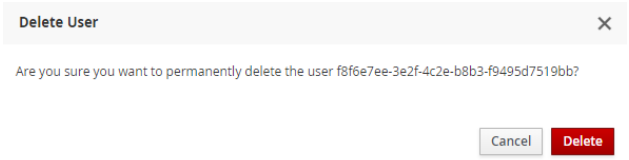


**Figure 33 Delete user confirmation box**

#### 5.2.2.4 User configuration

When we enter into a user profile, we access to a menu where we can edit different options of the user:

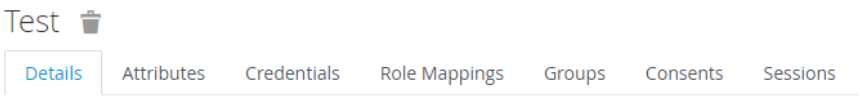


**Figure 34 User configuration menu**

Let's detail some of the most representatives:

- Details

  Here we can modify some general options of the user, basically the same we specified in the creation process of the user (email, first name, last name…)

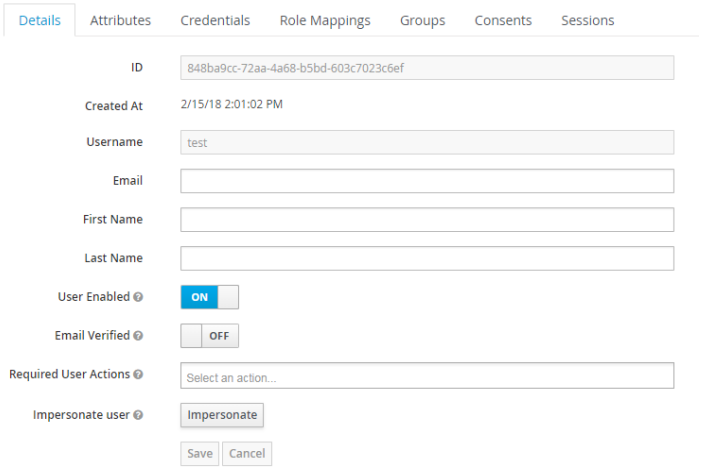


**Figure 35 User details tab**

- Attributes

  In this tab, we can manage specific arbitrary user attributes.

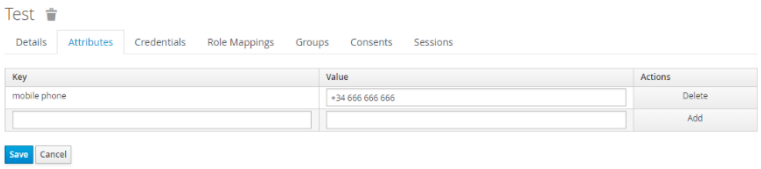


**Figure 36 User attributes tab**

- Credentials

  In this tab, we can manage the passwords and the OTP of the users.

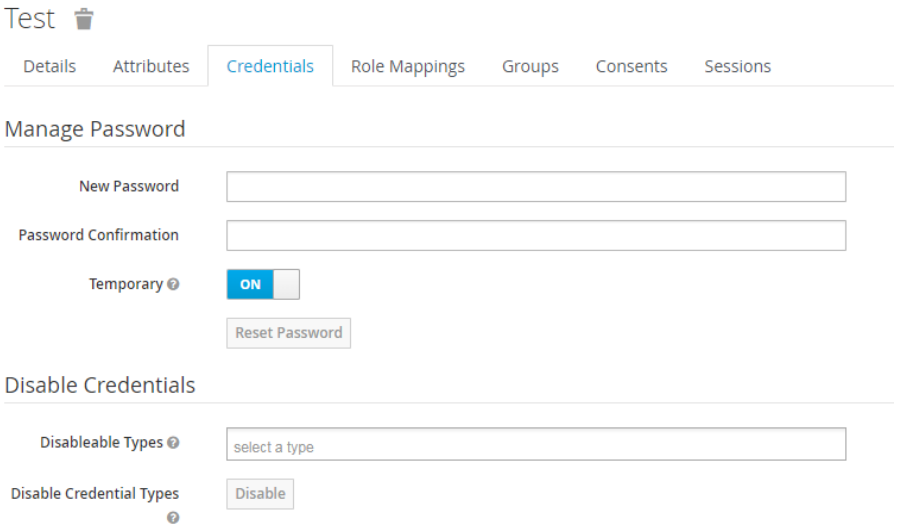


**Figure 37 User credentials tab**

- Role Mappings

  Here we can manage the different roles (a role is a predefined group of actions and behaviours to be applied to a set of users). Due to organisational and business requirements, sometimes it'll be helpful to group different users into a specific role.

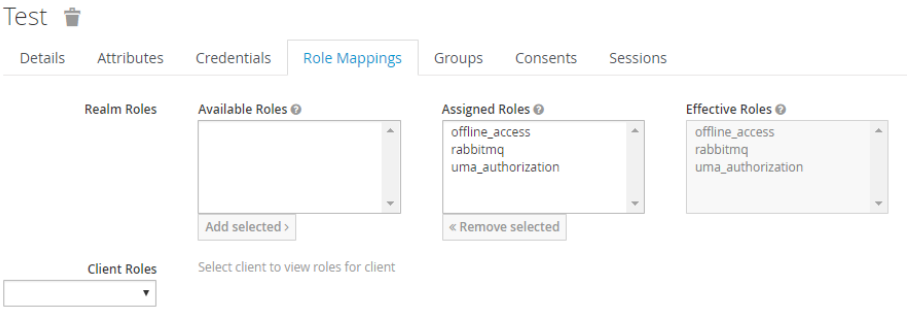


**Figure 38 User Role Mappings tab**

- Groups

  In this tab we can manage the grouping of users.

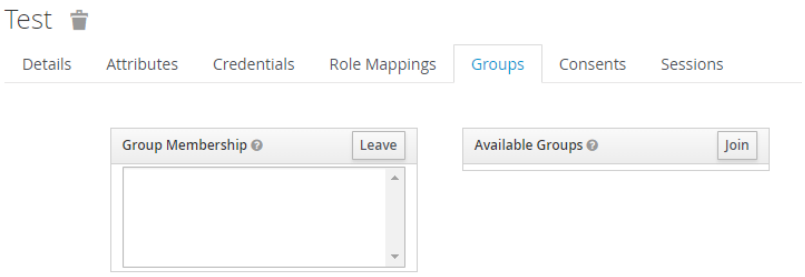


**Figure 39 User groups tab**

### 5.2.3 Authentication

#### 5.2.3.1 Password policies

When a new realm is created, there isn't password policies by default. So any user can create as short, as insecure, as complex password as they want. For production environments, simple settings are unacceptable and a set of password policies are needed.

For example, for the Composition-inter real, we click on the "Password Policy" option in the "Authentication" menu:

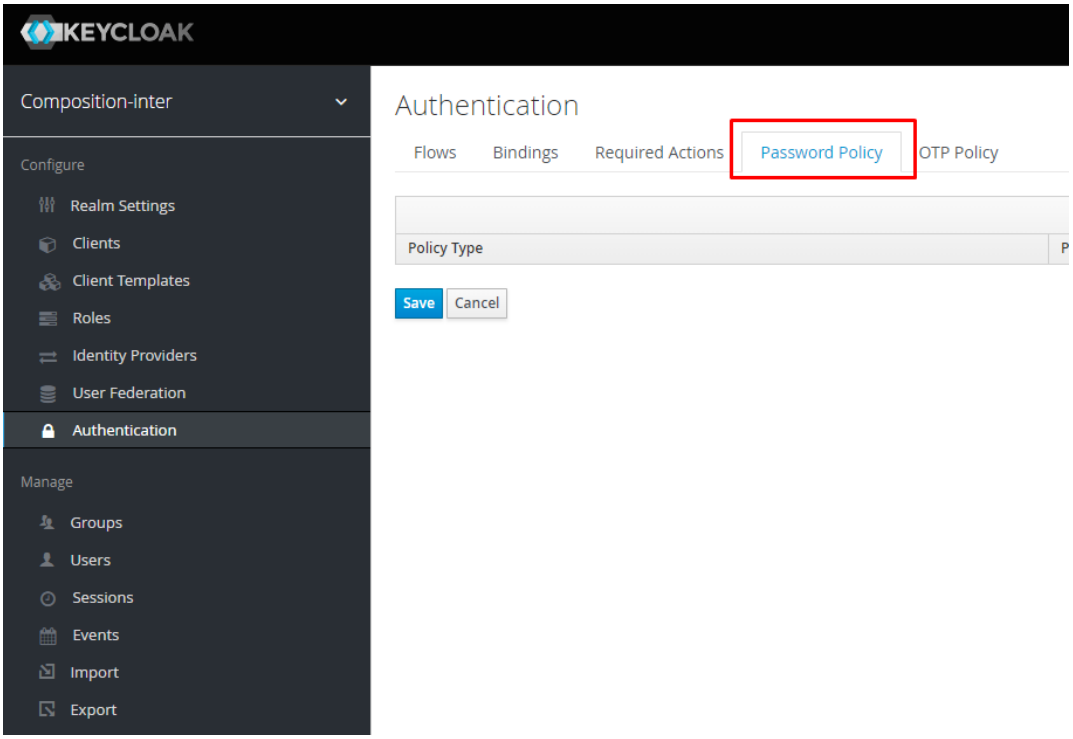


**Figure 40 Password policy page**

In the dropbox menu on the right, we can add new policies:

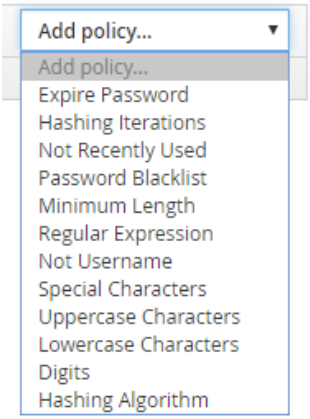


**Figure 41 Different password policies available**

After adding a new policy, or updating an existing one, user registration and a password update is required for the application of the new policy. Then, if we try to update the password according to the new policy, we should accomplish it:

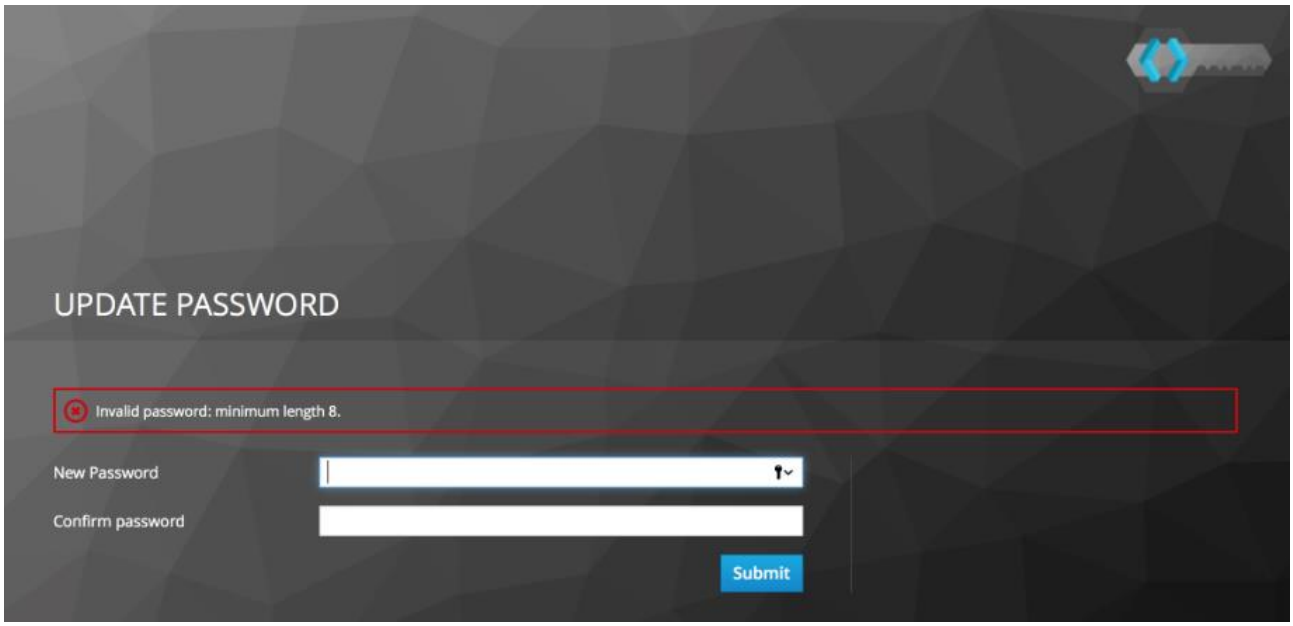


**Figure 42 Failed password policy**

#### 5.2.3.1.1 Password policy types

**HashAlgorithm**

Passwords are not stored as clear text. Instead they are hashed using standard hashing algorithms before they are stored or validated. The only built-in and default algorithm available is PBKDF2. See the Server Developer Guide on how to plug in your own algorithm. Note that if you do change the algorithm, password hashes will not change in storage until the next time the user logs in.

**Hashing Iterations**

This value specifies the number of times a password will be hashed before it is stored or verified. The default value is 20,000. This hashing is done in the rare case that a hacker gets access to your password database. Once they have access to the database, they can reverse engineer user passwords. The industry recommended value for this parameter changes every year as CPU power improves. A higher hashing iteration value takes more CPU power for hashing, and can impact performance. You'll have to weigh what is more

important to you. Performance or protecting your passwords stores. There may be more cost effective ways of protecting your password stores.

**Digits**

The number of digits required to be in the password string.

**Lowercase Characters**

The number of lower case letters required to be in the password string.

**Uppercase Characters**

The number of upper case letters required to be in the password string.

**Special Characters**

The number of special characters like '?!#%$' required to be in the password string.

**Not Username**

When set, the password is not allowed to be the same as the username.

**Regular Expression**

Define one or more Perl regular expression patterns that passwords must match.

**Expire Password**

The number of days for which the password is valid. After the number of days has expired, the user is required to change their password.

**Not Recently Used**

This policy saves a history of previous passwords. The number of old passwords stored is configurable. When a user changes their password, they cannot use any stored passwords.

**Password Blacklist**

This policy checks if a given password is contained in a blacklist file, which is potentially a very large file. Password blacklists are UTF-8 plain-text files with Unix line endings where every line represents a blacklisted password. The file name of the blacklist file must be provided as the password policy value, e.g. `10_million_password_list_top_1000000.txt`. Blacklist files are resolved against `${jboss.server.data.dir}/password-blacklists/` by default. This path can be customized via the `keycloak.password.blacklists.path` system property, or the `blacklistsPath` property of the `passwordBlacklist` policy SPI configuration.

#### 5.2.3.2 Authentication flows

An authentication flow is a container for all authentications, screens, and actions that must happen during login, registration, and other Keycloak workflows. If you go to the admin console Authentication left menu item and go to the Flows tab, you can view all the defined flows in the system and what actions and checks each flow requires. This section does a walk-through of the browser login flow. In the left drop down list select browser to come to the screen shown below:

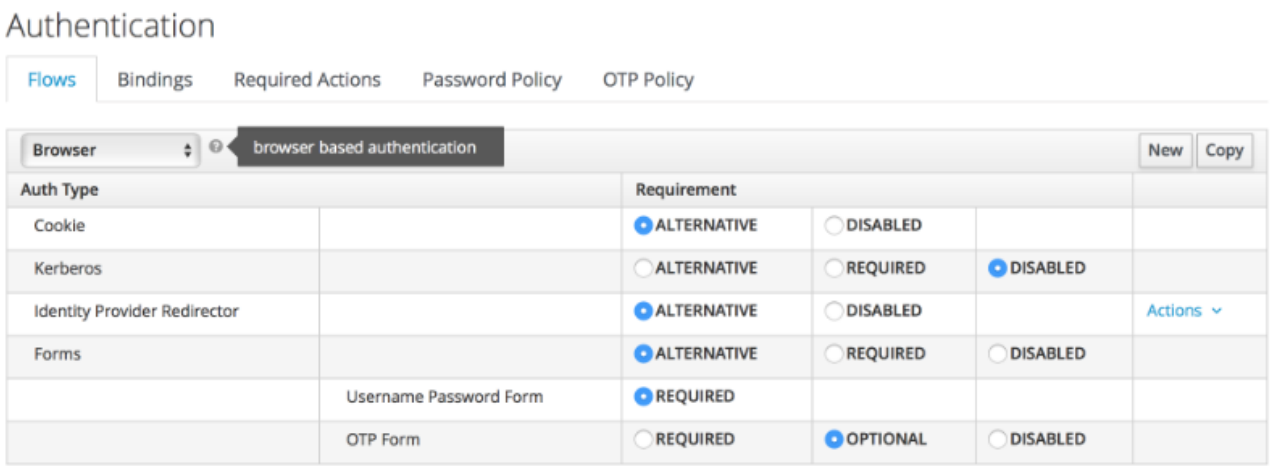


**Figure 43 Authentication flows**

The “Auth Type” column is the name of authentication or action that will be executed. If an authentication is indented this means it is in a sub-flow and may or may not be executed depending on the behaviour of its parent. The Requirement column is a set of radio buttons which define whether or not the action will execute. Let’s describe what each radio button means:

**Required**

This authentication execution must execute successfully. If the user doesn’t have that type of authentication mechanism configured and there is a required action associated with that authentication type, then a required action will be attached to that account. For example, if you switch OTP Form to Required, users that don’t have an OTP generator configured will be asked to do so.

**Optional**

If the user has the authentication type configured, it will be executed. Otherwise, it will be ignored.

**Disabled**

If disabled, the authentication type is not executed.

**Alternative**

This means that at least one alternative authentication type must execute successfully at that level of the flow.

Following the example of the Figure 43 Authentication flows:

- The first authentication type is Cookie. When a user successfully logs in for the first time, a session cookie is set. If this cookie has already been set, then this authentication type is successful. Since the cookie provider returned success and each execution at this level of the flow is alternative, no other execution is executed and this results in a successful login.
- Next the flow looks at the Kerberos execution. This authenticator is disabled by default and will be skipped.
- The next execution is a subflow called Forms. Since this subflow is marked as alternative it will not be executed if the Cookie authentication type passed. This subflow contains additional authentication type that needs to be executed. The executions for this subflow are loaded and the same processing logic occurs
- The first execution in the Forms subflow is the Username Password Form. This authentication type renders the username and password page. It is marked as required so the user must enter in a valid username and password.
- The next execution is the OTP Form. This is marked as optional. If the user has OTP set up, then this authentication type must run and be successful. If the user doesn’t have OTP set up, this authentication type is ignored.

## 5.3 RaaS – EPICA Integration

In this section, a first integration between the RaaS and EPICA will be described. The final, and improved, version of this integration will be presented in the next deliverable.

The RaaS is the component in charge of sending access control requests to EPICA. This can be done thanks to the API offered by EPICA itself, more precisely by the Authorization Engine, as described in Section 3.1.2.

A Keycloack authentication tokens will be inserted directly by the RaaS in the Authorization Header of the HTTP Request sent to EPICA. The idea is to encapsulate all the information needed by the latter, for performing the policy enforcement process, in the codified Keycloack token: EPICA will extract the needed data from the token, after decoding it, for building the XACML request, which will be sent to the Policy Decision Point. The final answer will be sent back to the RaaS.

Moreover, it will use the path parameters of the HTTP Request for inferring the location of the needed policies, to be used during the enforcement process. However, this solution is temporary, it is high probable that other methods for providing this information will be adopted, or simply fixed location in the local filesystem will be considered for storing the needed policies.

For checking how EPICA is dealing with Keycloack tokens, a simple test has been performed. Through the APIs offered by the Policy Administration Point, explained in Section 3.1.1, a basic XACML policy has been created, considering three attributes that could be extracted from the decoded token. These attributes were the following:

- Realm-access
- Company of the user
- Name of the user

After the policy creation, always through Swagger UI, the API offered by the Authorization Engine has been used, specifying the Keycloack token in the Authorization Header of the HTTP GET Request, “atos” as policy store, and “public” as resource id. The obtained result is shown in Figure 44.

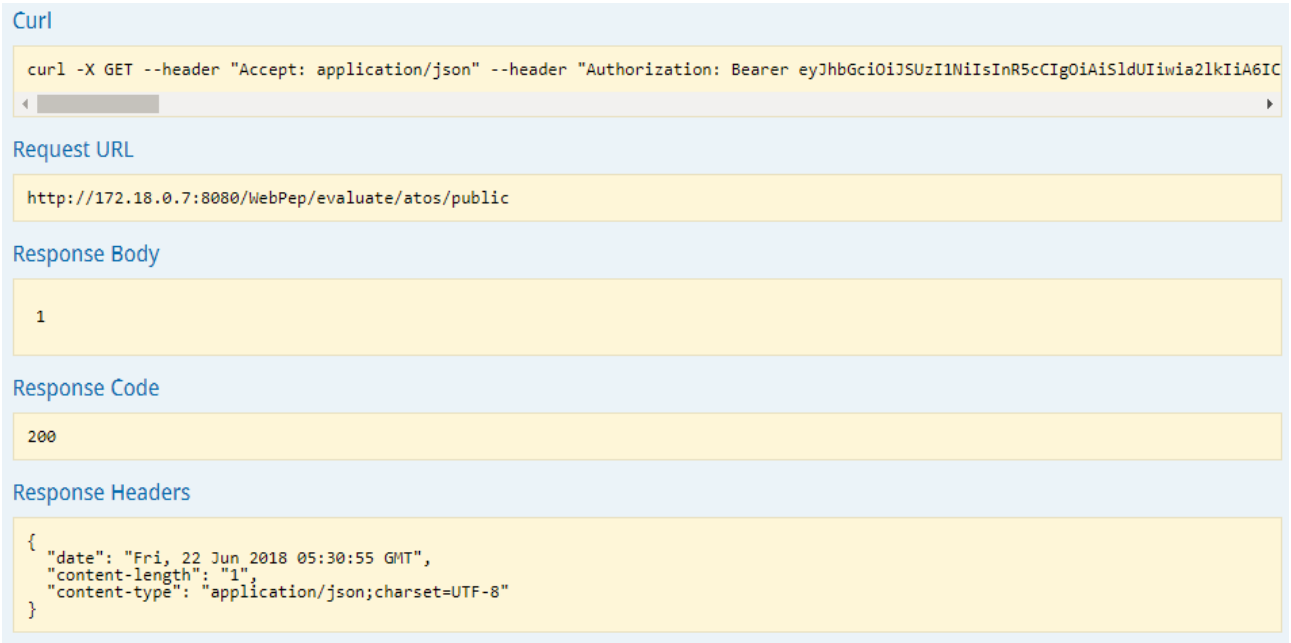


**Figure 44 Authorization Engine Test**

The response code confirms that the access has been granted by the Authorization Service. The IP address has been anonymised, as in the previous test in Section 3.1.1.

This little test concludes this section, as well as the overall part associated to EPICA, in this deliverable. Future works, especially related to EPICA, will focus on the definition of a specific set of policies and related attributes.

Moreover, the API offered by the Authorization Engine will be improved, and new ways for inferring the internal location of the policies when the request is received could be considered.

# 6 Conclusion

This first iteration of the deployment of the COMPOSITION Security Framework accomplishes the guidelines and requirements stated in D4.2 Design of Security Framework II. Due to is nature of a research project with high focus on market exploitation, and due to the living nature of the project, some changes would be required in the future, in order to secure the different components that are under development at this moment of the COMPOSITION framework (for example Marketplace Management portal).

Regarding the technological approach, all the technologies and security approaches are reused and widely tested due that this is not a security innovation project and no new technologies are tested, in order to preserve a security seamless approach and focus.

As well as the Security Framework is a very transversal components which provides a full set of functionalities based on component securing, this first deployment addresses requirements that interact with the rest of technical work packages, such as WP3 – Manufacturing Modelling and Simulation, WP5 – Key Enabling Technologies for Intra and Interfactory Interoperability and Data Analysis, and WP6 – Collaborative Ecosystem.

As main conclusion, we can state that this first iteration offers the main security functionalities based on authentication and authorization. Starting from this, the rest of the security functionalities and the pending integrations have a strong background for being achieved successfully in the lifecycle of the project.

# 7 List of Figures and Tables

## 7.1 Figures



## 7.2 Tables

# 8 References

(Resnick et al, 2002) P. Resnick and R. Zeckhouser, “Trust among strangers in internet transactions: Empirical analysis of eBay' s reputation system,” in The Economics of the Internet and E-commerce, Emerald Group Publishing Limited, 2002, pp. 127-157.

(Gutowska et al, 2009) A. Gutowska and A. Sloane, “Modelling the B2C Marketplace: Evaluation of a Reputation Metric for e-commerce,” in International Conference on Web Information Systems and Technologies, 2009.

(Sabater et al, 2001) J. Sabater and C. Sierra, “Regret: A reputation model for gregarious societies,” Fourth workshop on deception fraud and trust in agent societies, vol. 70, pp. 61-69, 2001.

(Llamazares, 2011) B. Llamazares, "On generalizations of weighted means and OWA operators." EUSFLAT Conf. 2011.

(Garcin et al, 2009) F. Garcin, B. Faltings, and R. Jurca, "Aggregating reputation feedback". Proceedings of the First International Conference on Reputation: Theory and Technology, Vol. 1, No. 1. 2009.